IJCSC International Journal of Computer Science & Communication Vol. 1, No. 2, July-December 2010, pp. 87-90

B+ Tree Based Indexing Scheme for FLOWR Queries on XML Databases

Anagha Vaidya[1] & Arpita Gopal[2]

[1,2]Sinhgad Institute of Business Administration & Research, Kondhwa(Bk.),Pune- 411048
Email: [1]anagha_vv@yahoo.co.in, [2]directormca_sibar@sinhgad.edu

ABSTRACT

XML is emerging as a de facto standard for information exchange over internet. To perform this task different languages are developed among them XPath and superset XQuery is popular language. XQuery is a strongly typed, functional language which supports the common processing and querying tasks. XQuery uses the label paths to traverse the irregular structure data. Without structural summary and efficient indexes, query processing can be quite inefficient due to exhaustive traversal on XML data. This paper presents an m-array based indexing scheme for efficient retrieval on FLOWR queries on XML databases. The proposed technique stores path dictionary, element dictionary and value node dictionary using B+tree structure. With help of this method we can access database quickly and effectively.
Keywords: FLOWR, XML, m-way search, B+tree

1. INTRODUCTION

The Extensible Markup Language (XML) is becoming the dominant standard for exchanging data over the World Wide Web. Due to its flexibility, XML is rapidly emerging as the de facto standard for exchanging and querying document on the Web. XML data is an instance of semi structure data and XML documents comprises hierarchically nested collections of elements represented as a tree, where element can be either leaf node or nested node. Due to simplicity in representation it is used in next generation web application including electronic commerce, intelligent web search. The different query languages are designed for that purpose. Lorel, Quilt, XML-GL, XPath, XQuery. Amongst XQuery is more popular language because it is functional language. It utilized regular path expression thus using convention tree traversal approach for same. It also specifies patterns of selection predicates on multiple elements that have some specified tree structured relationship. This relationship either parent child or ancestor-descendent relationship. Queries in XML make fundamental use of tree pattern for matching relevant portion of data in the XML database. The query pattern node labels include element tags, attribute-value comparison and string values and the query pattern edges are either parent-child edges.

Science XML data is a simple flat file. The data is scattered at different locations in the disk, processing XML queries may result in insignificant performance degradation. Hence we developed a new approach in which data is organized into different dictionary like path dictionary, element dictionary and value node dictionary index. With help of these dictionaries B+tree is developed. By using this B+tree query search performance is improved.

This paper is organized as follows: Literature Review is described in section 2.The structure of Proposed model is explained in section 3. Section 4 explain example of proposed technique. Querying on XML data is depicted in section 5. Conclusion and future scope is explain in section 6.

2. EXISTING WORK

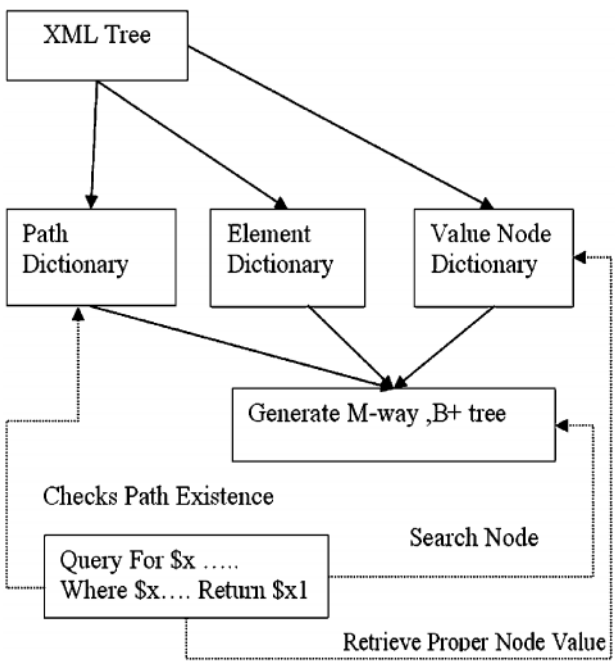
Depending on the context implementation of XQuery/XPath various implementation and optimization techniques are used. One of the techniques is index. Indexes are pre built on XML data which facilitate XML query processing. XML indexes are classified into two types - value index and structural index. In 'value index' indexes are created on data values in XML document and in structural index the index are created on the structural relationship. The major contribution on the indices technique is by Path index[10], Data guide[16], Index family[19], forward and Backward index[8], A(k)[17], D (K) indices[14] and APEXindex[2]. Index technique has problem of index size, index computational cost and index updatability. This problem can be overcome by 'numbering schema 'technique. Numbering Schema explain the hierarchical relationship presented in XML tree. It is classified into prefix labeling schema and range labeling schema. Dietz [13] introduced "Tree traversal schema", Li and Moon [15] proposed a numbering schema. Kimber [20] proposed "tree location address".

XML algebra is also used for query optimization. It provides semantic analysis of query and performs optimization. There are many different approaches are developed in designing algebra some of them XAL [4], Tree Algebra for XML (TAX)[7],Generalized tree pattern (GTP), Tree Logical Class (TLC), Nested XML Tableaux (NEXT), BiRD[5].

Relational approach, native approach, sequence approach are used for query processing and optimizing data. In relational approach XML data is converted into relational table and query processing is performed by using relational optimizer. Edge[3], Path Materialization algorithms[9] are different algorithm developed for same. In the native approach XML data are stored on disk in the form of inverted lists , sequences or trees and native algorithms are developed for query processing MPMGJN[22], Stack Tree [1], Holistic Join[18], Twing Stak solution are algorithm for native approach. ViST[6] and PRIX[12] technique introduced sequential approach . By using 'virtual tries' technique the data tree is encoded and query is optimized.

3. THE STRUCTURE OF PROPOSED MODEL

We now introduce an m-array based indexing scheme for efficient retrieval on FLOWR queries on XML databases. The proposed technique stores path dictionary, element dictionary and value node dictionary using B+tree structure. Overall structure of the technique is presented in following figure



The technique generates a Path dictionary, Element dictionary and value node dictionary. With help of element dictionary we generate B+tree structure which organized the all nodes of XML files into sorted order. M-way search can perform on this B+tree . The general algorithm is described as:

Step1: Generate the Path Dictionary.

Step2: Generate the Element Dictionary.

Step3: Generate Value node Dictionary.

Step4: Generate B+ tree.

Step5: Querying on XML database.

The detailed algorithm is as follows.

Step 1

Generate the Path Dictionary: It is a two-dimensional array as shown in table1. The first column of the table store Path name and in second column it store path expressions of XML document. When query is executed the existence of path is checked by this table.

Step 2

Generate the Element Dictionary: This is one dimensional character array and store the all element node from second level of the tree. In table2 all element nodes are store.

Step 3

Generate Value NodeDictionary: This is a two-dimensional array. As shown in table3, the first column indicate the new node value. Second column store element node string and third column store the corresponding values of the node. It also support in query 'return clause' for retrieval of XML node information.

Step 4

Generate B+ Tree: A B+ tree of order m is an m-way search tree that either empty or contain index node and data element node. The tree has 'm' sub tree and satisfy following property [23]

$$n\ ,\ A0(K1,\ A1),(K2,A2),\ldots\ldots\ldots.(Kn,An)$$

Where the Ai , 0 <= i <= n <m, are pointers to sub trees, and *Ki*, 1<= *i* <=*n* <*m* are keys be the format of some index node. All elements in the subtree Ai have key less than *Ki+1* and greater than or equal to *Ki* , 0 <= *i* < = *n*.

In proposed model 'm' is calculated from total number of element present in Element Dictionary. Key values are values of Element Dictionary. Element nodes of the b+tree are the address of the Value Node Dictionary . It store the entire value of the node.e.g. *v*[0]..*v*[2] store all the value of a node . New node identification is perform by New node column. If it contain "*Y*" value means from that node a new node is start.

Step 5: Querying on XML Database

The technique supports FLOWR queries on XML databases. The algorithm is:-

1 : Retrieve the query path from for clause.

2 : Search the "query path" into Path dictionary.

3 : If path exist

3.1 : Identify the predicates. i.e from where clause identify the search element

3.2 : From that predicate retrieve the element node from b+ tree.

3.3 : The Element node provides the address of "Vale node table".

3.4 : From Value node table get the value node information.

3.5 : From "return clause identify" the return node string . Match this string with token column in value node table. And the retrieve respective node.

3.6 : Print the value .

4 : Else print the message "node doesn't exit'

4. **Example of proposed technique:**

Let's consider XML file example :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
    <book category="computer">
            <title lang="en">MIS</title>
            <author>Arpita </author>
            <year>2007</year>
            <price>300.00</price>
    </book>
    <book category="XML">
            <title lang="en">XQuery Processing </
title>
            <author>
                    <fname>Anagha</fname>
                    <lname>Vaidya</lname>
            </author>
                    <year>2003</year>
                    <price>149.99</price>
            </book>
 </bookstore>
```

Step1: Generate Path Dictionary:

Table1
Path Dictionary

Path Name	*Path address*
P1	/bookstore/book
P2	/bookstore/book/title
P3	/bookstore/book/author
P4	/bookstore/book/year
P5	/bookstore/book/price
P6	/bookstore/book/author/fname
P7	/bookstore/book/authtor/lname

Step 2: Generate Element Dictionary

Table 2
Element Dictionary

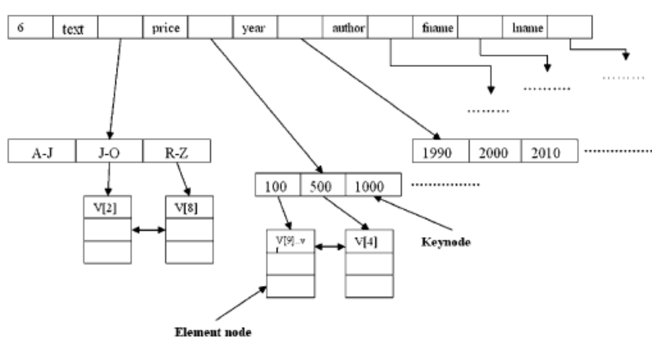
title	price	year	Author	Fname	lname

Step 3: Generate Value Node Dictionary

Table 3
Value Node Dictionary

New node	*Token*	*Value information*
Y	year	2003
N	title	MIS
N	lname	Vaidya
Y	year	2009
N	price	300
N	fname	Arpita
N	title	XQuery Processing
N	price	49.99

Step 4: Generate B+tree



B+ Tree for Element Node

Step 5: Querying on XML database:

Let's consider example

Query : For $x in (bookstore.xml)/bookstore/book

where $x /price >30

return $x /title

The for clause selects all book elements. The path address is "bpookstore/book/" which is store into a variable "$x". According to indexing method the value of $x is check in Path Dictionary . If Path exist then read the variable of " where" clause. In our example the "Price" element will be read . Then in the b+tree, search is performed on the node 'Price' and respective "Element Node" array is retrieve. The element nodes store the address of Value Dictionary Node. E.g. in example price value which is less than 30 the information is store into 'Value Dictionary Node" v[9] till v[11], v[13] till v[15]. The return clause specifies name of return node. In out example we want to return "title". This string will match with token string of retrieved "Value Dictionary Node" node. The proper node information will be display.

6. CONCLUSION AND FUTURE WORK

In this paper we have proposed a novel algorithm for storing XML data file. With help of this method we can access database quickly and effectively. Method handles simple XQuery , range query processing also . In this algorithm do not consider of handling the 'ordered clause' and nested queries In future we come up with a set of optimization methods handle these and improve performance.

REFERENCES

[1] C. Zhang, J.F. Naughton, D.J. DeWitt, Q. Luo, and G.M. Lohman, [2001]: "On Supporting Containment Queries in Relational Database Management Systems," *Proc. 20th ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '01)*.

[2] C.W. Chung, J.K. Min, K. Shim [2002] "APEX : An Adaptive Path Index for XML data", ACM SIGMOD pp. 121-132.

[3] D. Florescu and D. Kossmann ,[1999]: "Storing and Querying XML Data Using an RDMBS," *IEEE Data Eng. Bull.*, **22**, pp. 27-34.

[4] F. Frasincar, G.-J. Houben, and C. Pau. XAL [2002] : "An Algebra for XML Query Optimization", *In ADC 2002*, Melbourne, Australia, ACS.

[5] F. Weigel, K.U. Schulz, and H. Meuss, [2005] : "The BIRD Numbering Scheme for XML and Tree Databases—Deciding and Reconstructing Tree Relations Using Efficient Arithmetic.

[6] H. Wang, S. Park, W. Fan, and P.S. Yu,[2003]: "ViST: A Dynamic Index Method for Querying XML Data by Tree Structures," *Proc. 29th ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '03)*.

[7] H.V. Jagadish, L.V.S. Lakshmanan, D. Srivastava, and K. Thompson, [2001]: "TAX: A Tree Algebra for XML," *Proc. Eighth Int'l Workshop Databases and Programming Languages (DBPL 1)*.

[8] Kaushik R., Bohannon P., Naughton J. F., Korth H. F.[2002] : *Covering Indexes for Branching PathQueries*, ACM SIGMOD.

[9] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura, "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases," *ACM Trans. Internet Technology*, **1**, pp. 110-141, 2001.

[10] McHugh J., widom J. , Abiteboul S.,Luo Q.., Rajaraman A.[1999] : Index Semistructure Data Technical Report, Srandford University.

[11] N. Bruno, N. Koudas, and D. Srivastava, [2002]": Holistic Twig Joins: Optimal XML Pattern Matching," *Proc. 21st ACM GMOD Int'l Conf. Management of Data (SIGMOD '02)*.

[12] P. Rao and B. Moon,[2004]: "PRIX: Indexing and Querying XML Using Prufer Sequences," *Proc. 20th IEEE Int'l Conf. Data Eng. (ICDE '04)*.

[13] P.F. Dietz, [1982] : "Maintaining Order in a Linked List", *Proceeding of the 14th Annual ACM Symposium on Theory of Computing*, pp. 122-127.

[14] Q. Chen, A. Lim, K. Ong, and J. Tang,[2003] "D(k)-index: An Adaptive Structural Summary for Graph Structured Data", *Proceedings of the ACM SIGMOD*, pp. 134–144.

[15] Q. Li & B. Moon, [2001]: "Indexing and Querying XML Data for Regular Path Expressions", *Proceeding of 27th VLDB Conference*, pp. 361-370.

[16] R. Goldman & J. Widom,[1997] "Data Guides : Enabling Query Formulationand Optimization in Semistructured Database", *Proceeding of VLDB* , pp. 436-445.

[17] R. Kaushik, D. Shenoy, P. Bohannon, E. Gudes, [2002] "Exploiting Local Similarity to Efficiently Index Paths in Graph-Structured Data", *Proceeding of Int. Conference on Data Engineering*, pp. 129-140.

[18] S. Al-Khalifa, H.V. Jagadish, N. Koudas, J.M. Patel, D. Srivastava, and Y. Wu[2002]: , "Structural Joins: A Primitive for Efficient XML Query Pattern Matching," *Proc. 18th IEEE Int'l Conf. Data Eng. (ICDE '02)*.

[19] T. Milo & D. Suciu, [1999], "Index Structures for Path Expression", *Proceeding of 7th Int. Conference on Database Theory*, pp. 277-295.

[20] W.E Kimber,[1993]: "HyTime and SGML : Understanding the HyTime HYQ Query Language", Technical Report Version 1.1, IBM Corporation.

[21] XQuery 1.0: An XML Query Language W3C Recommendation 23 January 2007.

[22] Z. Chen, H.V. Jagadish, L.V.S. Lakshmanan, and S. Paparizos,[2003]: "From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery," *Proc. 29th Int'l Conf. Very Large Data Bases (VLDB '03)*.

[23] Horowitz. Sahani. Anderson-Freed "Fundamentals Of Data Structures in C" University Press 2008.