XQUERY : Efficient searching and indexing Algorithm Based on Multi- tries

Anagha Vaidya[1] and Dr. Arpita Gopal[2]

Asst. Professor Director – MCA
Sinhgad Institute of Business Administration & Research,
Kondhwa(Bk.),Pune- 411048
[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. XML file is plan text file and self explanatory therefore XML users are increase dramatically. There are two type of groups who has interest on XML utilization effectively, the first group is mainly interested in XML as a humanly consumable exchange format. They are looking up for tools that allow easy manipulation and transformation of small XML document. Second community is mainly interested in XML as a data storage format. Their primary focus is on design of query languages for storage effectively. This data-oriented paradigm has taken more efforts in query evaluation. XML data doesnot have fix schema, managing the document is difficult job . It is therefore major challenge for database community to design query languages and storage methods that can retrieve data effectively. There has been a growing need for developing high-performance techniques to query large XML ..

The proposed technique summarized XML document into an 'x-store' , which store xml document distinct node :- element node, attribute node and value element node value, and path information. The unique storage address is generated for each node and it is indexed on path name.. Then we come up with the novel method of query searching which utilize multi-way tries to answer the query effectively .
Keywords- XML, XQuery, XPath, XML-XDM

Keywords : *XML, XQuery, XPath, XML-XDM*

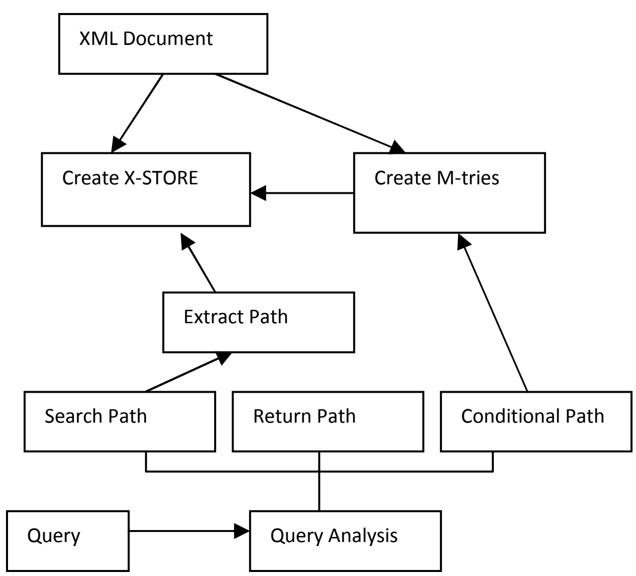
1. INTRODUCTION

Over the past decade the expansion of internet and complex web application such as e-commerce, e-learning requires data storage of different format. Since February 1998, the eXtensible Mark-up Language (XML) has become the standard medium for data representation and exchange over the web. As a result, the size and the number of XML's users increased dramatically. There are two type of group who has interest on XML utilization effectively, the first group is mainly interested in XML as a humanly consumable exchange format. They are looking up for tools that allow easy manipulation and transformation of small XML document. Thus optimization of the query not seems very useful. But second community is mainly interested in XML as a data storage format. Their primary focus is on design of query languages for storage effectively. This data-oriented paradigm has taken more efforts in query evaluation. In this context many query languages such as Lorel, Quiit, XQL, XML-QL, XPath and XQuery have been developed. Out of this two most popular languages are XPath and superset XQuery.

XML queries can classify[12] in three ways :- Tree structure, Starting node, Node type . Tree structure query is further classify into simple query and branch query . Starting node query classify into 'total match' and 'partial match' query. Node queries are called as content based query because they check the element and attribute. For execute[17] any type query it is parsed analyzed and execute, it returns one output node a single tree or multiple set of a tree called as "twig" which is a sub set of xml tree. To get result XML data must be store structurally i.e parent child, ancestor-descendent relationship properly. The main problem[16] is storing these structure effectively such that storage size is small and query performance time is less.

In literature , different techniques have been design to process query one of them is indexing. This paper tries to address the query processing problem by developing 'path-encoding schema and searching algorithm. We are storing XML file in such a way that it maintain structural relationship and search algorithm works on 'predicate node' and retrieve data on least I/O cost.. The paper is organized as follow: section 2 we introduced structure of proposed technique. In section 3 Algorithm , section 4 Query processing model Section 5 Explanation of model with help of example section 6Background study , section 7 conclusion and future work.

2. STRUCTURE OF PROPOSED TECHNIQUE



Step1:- Parsed XML document

For parsing XML files, two dominant models exist: DOM (**D**ocument **O**bject **M**odel) and SAX (**S**imple **API** for **X**ML parsing). We used SAX parser for our implementation.

Step 2:- Generate data structure

a)Create X-storage :- XML document parsed and each xml nodes are organized into a bucket (Flag, Path-Name, Type of node, Value). The 'Flag' column store a node number , which is similar as floating point schema[6]. number help in retrieving node for answering query. The 'Path Dictionary' store the path name of each XML node. 'Type of node' column store type of XML node –whether it is attribute node, element node, value node. If it is value node then exact value is printed from value column. If it is element node then it store 'null' value . The 'flag' column decide how many rows are display for respective query. Node address are store in node-address table . X-storage is hash index on 'Path Dictioanry' and for every pah-value hash values are store into h-table

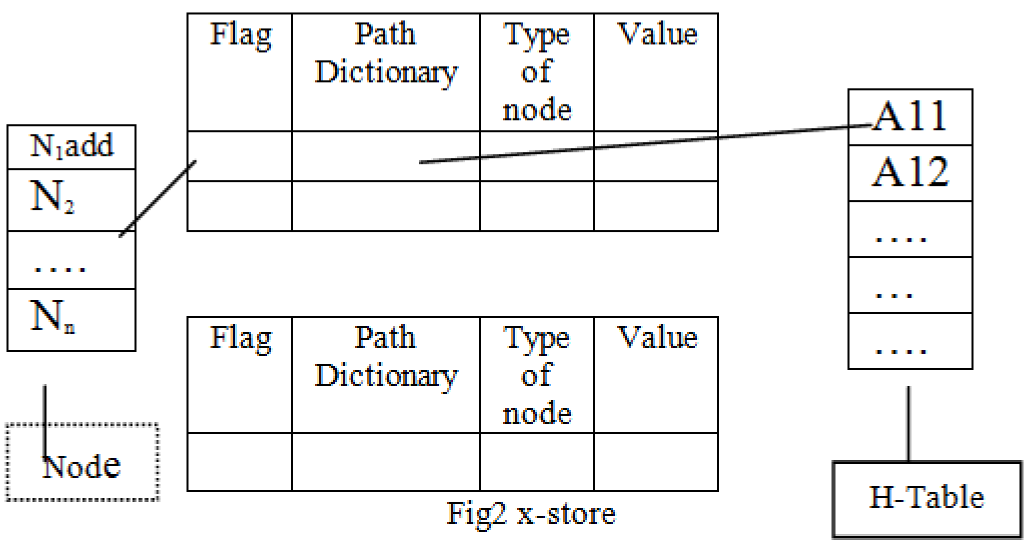


Fig2 x-store

Step3 Create Multiwaytries

A tries is a tree of degree m>=2 in which the branching at any level is determined not by the entire key value but only a portion of it. The tries contain two type of node: element node and branch node. Element node has only data field and branch node contain pointers to sub tree. The 'multiway tries' help in searching a node. 'Multiway tries' are constructed on the 'Value Node' of xml file i.e branch node are generated on "Value Column' . In element node store the address of respective X-store rows.. E.g. 'name' is a value node . It appear in xmlnode1 and xml node2. The data filed store address of xmlnode1 and address of xml node2

ALGORITHM

a) *creatinh m-way tries*

1. char ch, int index;

2. M1 = Creation of m-way tries
3. Define triespointerarray[26]
4. Defien element node which contain the address of table X-store (start node address, end node address)
5. Read the Key values .(all value node of the XML file)
6. ch = sample(total number of key value) ; // perform samples of x for the branching at 'i' level
7. index =getindex (keyvalue, ch) // function return integer value of character for i^{th} index of m-way tries.
8. if (m-way[index]== NULL)
    a. add element node
    b. exit
9. Else
    a. If (M1->ptr) == NULL
        i. M2= Create new(m-way tries);
        ii. M1->ptr = M2;
        iii. Go to 8;
    b. Else
        i. M1= M1 ->ptr;
        ii. Goto step 9;

b) Algorithm for sampling :-

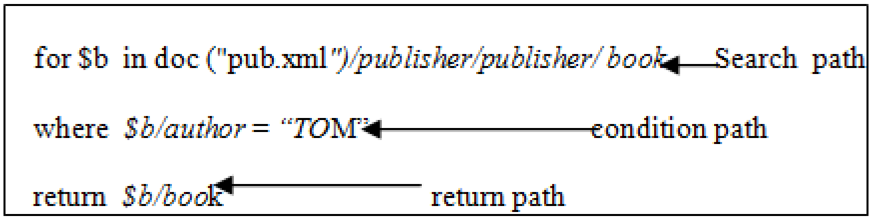
```
int    sample(int N)
    {  int   i;
            level = Enter the level
            if (level == even)
                    i  = (level /2);
             else
                i=  N -  (level -1) /2;
            return (i);
    }
```

c) get index:-

```
int    getindex(key value, i)
    {     char ch, int j;
          ch =  keyvalue[i];
          j   = convert ch into integer
          return(j);
    }
```

Step 4 :- Query Analysis :

Query is parsed and validate. These tokens are provided to 'XQuery Analyzer '. It analyze token and create separate token path : "Search Path", "Conditional Path" and " Return Path". This is illustrate with simple query below:-



processsearch Path(): Returns the elements in the path specified in the FOR clause.

processConditional Path(): Returns the subset of elements returned by *processForPath()* satisfying the condition in the WHERE clause of the XQuery file.

processReturnPath(): Returns the elements (usually descendants of the elements returned by *processconditional()*) specified by the RETURN clause.

QUERY PROCESSING

Query is parsed and validated . From "Query Analyzer class" process different path i.e 'search path', 'condition path' and 'return path'. First step 'Conditional path is extracted and extract predicate node we called it as 'conditional node' and it is input to "Query evaluation classs" . The conditional node is search into a first tries. First tries is constructed on value node thus respective node is link . Link to next tries for further search. Search exact value and identify desire node. We reach to leaf node i.e element node. The element node of the multi way tries provide address of the bucket. Fetch the bucket from x-store desire node is available. Read 'Return Path'. Generate hash value of respective path. By using hash-table i.e. h-table search node in extracted node only.. Take out the row from the x-store. Read the flag value (start node position, end node position) of the row and 'type of element' . If 'type of element' is 'element node type ' then read all rows from start node position till end node position and display result. If it is value node display value.

Step 1:- Query Formation
- Read the query
- Check for grammatical error
- If no error found then

  a. Create three paths

    i. Search path – represent search path

    ii. Condition Path – represent conditional node

    iii. Return Path - represent return path

  Call Query Evaluation procedure

Step 2:- Query Evaluation-
- If 'Search-Path' Exist in path –Dictionary
  - Key-value-> conditional value
  - Create m-way tries
  - Call search() algorithm
- Else
  - Print element is not

Step 3 algorithm for Search Element

```
search()
       {
           key value -> read the conditional path
         i-> getindex(key value, index);
         if (m-way tries = = element node)
                    retrive the bucket address
                    display(x-store address)
         else
                    m-way tries -> m-way tries ->next;

        }
```

Step4 :- Display Function :-

```
        display( x-address)
           {  search-string ->  return-path;
               search_string (x-address)
         If (element node)
                     while(!endpoition)
                       {read  row
                        display row
                       }
                    else display row content
           }
```

PERFORMANCE ANALYSIS

We develop java code for above algorithm and construct four classes : “M-way tries”,”Query Analyzer”, “Query evaluation” “Node Searching” in Java and work with following set of queries on “Student Dataset”-student.xml. The student.xml consist of 12 number of document , 47 maximum fan out element and 3 depth of element and total number of node 2990. Sample XML file is represented in fig 3. Different axes XQuery is used they are listed on table1 . Result is presented in graph

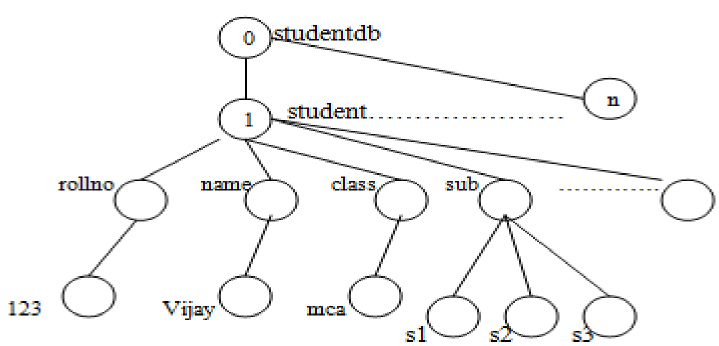
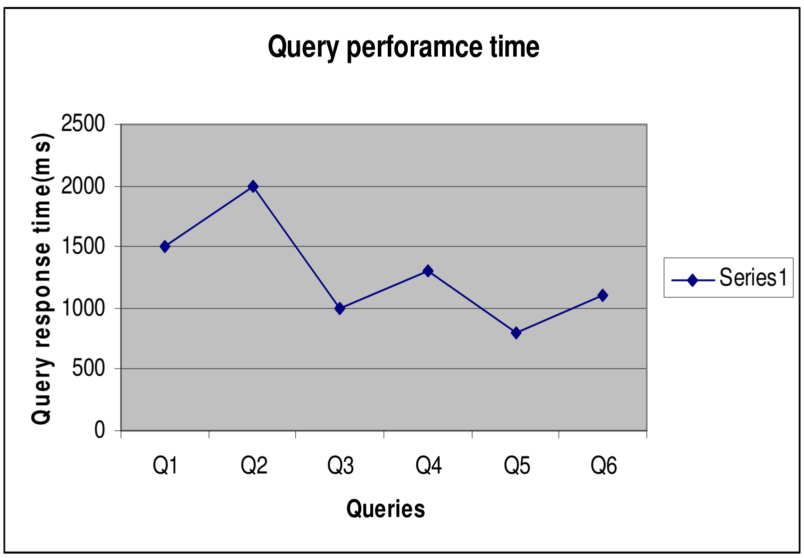


fig 3 XML file structure

Table 1 Test queries on student file

Q1	/studentdb/student::*
Q2	for $s1 in ("student.xml")/studentdb/student where $s1/class="mca" return $s1
Q3	for $s1 in ("student.xml")/studentdb/student where $s1/class="mca" return $s1/name
Q4	for $s1 in ("student.xml")/studentdb/student let $x := $s1/name/text() order by by $x return $s1
Q5	for $s1 in("student.xml")/studentdb/student where $s1/sub != "s1' return $s1/rollno
Q6	for $s1 in("student.xml")/studentdb/student where $s1/rollno!= 123 return $s1/rollno

We observe following result:-



Existing Work

Indexing store the structural relationship of all the elements and attributes. These indexing technique have been group into four category : node encoding, path encoding, sequence-based , feature-based indexing . Querying XML data is nothing but find out the proper path These path is identify by the Query processors by identify xml elements and attributes from schema.. Main challenge for it is path identification from irregular XML document to retrieve proper result because query output may be a simple node or twig pattern. We study only structural indexing .

According to our knowledge very first techniques for Xpath expression evaluation was developed by McHugh et in 1999[2] along with various structural and value index. In early development of XML indexes are build on path summary starting from root node to respective node . DataGuid [1] are general path indexed that summarize all paths in the data graph that star from root. It does not preserve hierarchical relationship among individual nodes. Thus there is no path information between any node. To solve query 'DataGuid' need to traverse the original data source. This is applicable only simple path expression and not feasible for multiple path expression. To overcome the limitations of 'DataGuid ' Millo and Suciu[3] introduced Index family consists of 1-index, 2-index and T-index. 1-index restricted to simple path queries only and work similar to 'DataGuid' . 2-index locate all pair of ancestor descendent element that are linked by specific sequence tags. This are useful for the query which has branch path expression. But 2- index grows very large. In T- index appropriate templates have been built in advance. It reduced size of index making sure that every node in he index document appear exactly once in the index . A major drawback of this is the size of index may still grow very high unstructured data.

Next generation indexes are build on local similarity to reduced size of index graph. Apex [5] index utilizes only frequently used paths by applying sequential pattern matching. It consists of two structure graph structure and hash tree. Kaushik et[7] introduced A(K) index , which groups nodes on k-bisimilarity. The parameter K controls the convergence and size of A(K) index.It work accurately if the length of path expression is less than or equal to k. Chen *et al* [9] proposed D(k) index which is an adoptive structure summary for general graph-structure data. It is also based on bi-similarity , It is flexible and smaller size as compare with A(K) index but have major drawback; over-refinement of irrelevant index , data nodes and over-qualified parents.

Another group covers the forward and backward path to support all queries for branch path expressions effectively. Forward and Backward Index[4], used data structure "Disk based clustering tree "so called Forward-Backward (F&B)indexes which can be used as covering indexes for XPath expressions using the correlated label paths, twig patterns for managing XML File .Usually F & B indexes are based on structures similar to DataGuides . in practice the indexes based on structural summaries tend to be very large. Their size often almost reaches the size of indexed data. To overcome this drawback Kaushik et al.,[8] proposed BPCI , an index schema which restricts the class of queries that can answered by eliminating branch path expressions that are estimate less importance to reduced index size

All above method focus on storage space efficiently and store path information as structure summary , next generation index technique compress XML tree . Wang et al [10] proposed index technique called VIST .The XML data and XML queries are transformed to structure encoded sequences and virtual suffix trees are used to speed up the pattern matching process. VIST also unifies structural indexes and values indexes into a single index, hence it is more efficient than pure value or structure indexes. The same but more advanced concept was used for PRIX index [11]. Every XML document in the database is transformed into a sequence of labels by Pr¨ufer method that constructs a bisection between the tree and a sequence. The query twigs are also transformed to such sequences and subsequence matching is used to find the twig patterns in the XML trees.

Kiss and Anh [13] combine both structural and tree structure index to accelerate the query processing. The structural index is simulated by 1-index and the efficiency of the query evaluation is improved by using tree structure index on the 1-index. Zhang *et al.* [14] propose FIX, a feature-based indexing technique, based on spectral graph theory. For each twig pattern in XML document, they calculated a vector of features based on its structural properties and stored in a multidimensional index tree as a key. Given a query, its feature vector is first calculated and looked up in the index. Then a further refinement phase is performed to fetch the final results. Jiang *et al.* [15] introduce GString, a novel sequencing method to capture the semantics of the underlying data graph. Meaningful components of the graph structure are located and used as the basic units in sequencing. This technique reduces the size of resulting sequences and also enables semantic-based searching.

Conclusion and Future Work

In this paper, we propose model for answering 'FLWOR ' types of queries by investigating the classical index technique by taking advantage of floating point schema [6].

We design indexing technique which can avoid redundant searching on XML storage. The structure is simple and able to support queries with combination of parent-child and ancestor-descendent edges and avoid multiple join. We proposed search algorithm will search the query result minimal I/O cost and speed up query evaluation process.

There are several avenue for future work , such as how to store XML file dynamically so that when new node is added or deleted re-indexing is not required and flag design must be differentiate attribute and value node. It designing compression technique for searching node thus search can perform more efficiently and design optimization method for the best performance.

References

[1] R. Goldman, and J. Widom, "Data Guides : Enabling Query Formulation and Optimization in Semistructured Databases", Proceedings of the VLDB, 1997, pp. 436-445.

[2] J. McHugh, and J. Widom, "Query Optimization for XML", Proceedings of VLDB, 1999, pp. 315-326.

[3] T. Milo, and D. Suciu, "Index Structures for Path Expression", Proceedings of the ICDT, 1999, pp. 277-295.

[4] S. Abiteboul, P. Buneman, and D. Suciu, "Data on the web: from relations to semistructured data and XML", Morgan Kaufmann Publishers, USA, 1999.

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

[6] T. Amagasa, M. Yoshikawa, and S. Uemura. QRS: A Robust Numbering Scheme for XML Documents. In Proc. of ICDE, pp705-707, 2003.

[7] R. Kaushik, D. Shenoy, P. Bohannon, and E. Gudes, "Exploiting Local Similarity to Efficiently Ordex Paths in Graph-Structured Data", Proceedings of the ICDE, 2002, pp. 129-140.

[8] R. Kaushik, P. Bohannon, J.F. Naughton, and H.F. Korth, "Covering indexes for branching path queries", Proceedings of the ACM SIGMOD, 2002, pp. 133-144.

[9] Q. Chen, A. Lim, K. Ong, and J. Tang, "D(k)-index: An adaptive structural summary for graph structured data", Proceedings of the ACM SIGMOD, 2003, pp. 134–144.

[10] Wang H., Park S., Fan W., Yu P. S.: VIST: A Dynamic Index Method for Querying XML Data by Tree Structures, ACM SIGMOD 2003, June 9-12, San Diego, CA, 2003

[11] Rao P., Moon B.: PRIX: Indexing And Querying XML Using Pr¨ufer Sequences, In proc. Of ICDE 2004, Boston, USA March 2004

[12] Barbara Catania and Anna Maddalena Athena Vakali "XML Document Indexes: A Classification" Published by the IEEE Computer Society 1089-7801/05/$20.00 © 2005 IEEE

[13] A. Kiss, and V.L. Anh, "Combining Tree Structure Indexes with Structural Indexes in Query Evaluation on XML Data", Lecture Notes In Computer Science, 3631, 2005, pp. 254-267.

[14] N. Zhang, M.T. Ozsu, I.F. Ilyas, and A. Aboulnaga, "FIX: Feature-based Indexing Technique for XML Documents", Tech. Rep. No CS-2006-07, University of Waterloo, 2006

[15] H. Jiang, H. Wang, P.S. Yu, and S. Zhou, "GString: A Novel Approach for Efficient Search in Graph Databases", Proceedings of the ICDE, 2007, pp. 566-575.

[16] Mohammed Al-Badawi , Dr. Siobhán North, Dr. Barry Eaglestone "Indexing XML Databases: The Classifications, Problems Identification and a New Approach " technical report 2007 The University of Sheffield Department of Computer Science

[17] XQuery 1.0: An XML Query Language W3C Recommendation 23 January 2009