ISSN:2320-0790

An International Journal of Advanced Computer Technology

A Survey of Indexing Techniques for Xml Database

Atul D. Raut[1], Dr. M. Atique[2]

[1]Deptt. of I.T. J.D.I.E.T. Yavatmal
[2]PGDCSE SGBAU Amaravati

Abstract: XML data is stored in the form of a text document. Hence different applications can share the XML data. It is platform independent i.e. it can be migrated to any operating system without any changes. Moreover it is self describing and extensible as a result of which it has became an emerging standard for data exchange and storage over the web. With such growing presence of XML in database technology there is a growing need to develop efficient storage and indexing techniques for querying large repositories of XML documents. In this paper we review some of the important indexing techniques for XML and compare these techniques on the basis of key factors for storing and querying XML documents. The real work in indexing of XML started with the introduction of inverted list to store XML data. This inverted list with different query algorithm could answer a variety of Xpath queries but with some limitations. Inverted list was further improved by the concept of structure index which is a compact summarization of tree representation of XML document. Very recently efficient indexing techniques were implemented. These recent techniques made use of efficient data structures such as the B^+ tree for storing and querying the XML data. Experimental results of these techniques show a significant improvement in performance over the inverted list technique.

Keywords: inverted list; structure index; xreal; xcdsearch; dewey id

I. INTRODUCTION

XML is an important development for database applications since the relational model. It is self describing and extensible as a result of which it has became an emerging standard for data exchange and storage. With such growing presence of XML in database technology there is a growing need to develop efficient storage and indexing techniques for querying large repositories of XML documents. Indexing techniques used for relational database cannot be used directly for XML since XML data is ordered where as the relational data is unordered. Moreover XML contains structure in addition to data. The presence of structure makes the task of indexing much more difficult as compared to relational database. The most important factor for any efficient querying system is the time required to get result. This response time depends upon the intermediate result size, index size, the amount of I/O required, compression/decompression time etc. The response time can be significantly reduced with the support of efficient indexing and storing technique for XML data[1]. XML documents can be represented with the help of an ordered labeled tree. W3C query languages Xquery and XPath specify patterns of selection predicates on multiple elements that have some specified tree structured relationship [2] [3]. For example the Xquery expression book [title = 'XML' // author [fn = 'jane' AND ln = 'doe'] searches for the author element having child or sub element as fn with content jane and ln with content doe (parent-child relationship) and all author should be descendants of book element (ancestor – descendant relationship) having child element title with content 'XML'Page Layout

Thus it is clear that any XML query has two major components the structure component and the keyword (data information) component. In earlier techniques an XML query can be answered by finding all occurrences of such twig pattern and then stitching (joining) together these matches. Twig pattern matching can be achieved by decomposing the twig into a set of binary structural (parent child and ancestor – descendant) relationship between pairs of nodes [4] [5] [6]. For joining the matches different join algorithms were proposed and implemented. These algorithms were I/O and CPU optimal but

generated large intermediate results, which eventually increased the query response time. These earlier techniques and the recent indexing techniques are summarized below.

II. RELATED WORK

Till date several techniques have been implemented for querying XML documents using different indexing techniques by different researchers. These early techniques can be broadly classified into following types.

a. By traversing the tree or its compressed representation.
b. By using IR style processing using inverted list.
c. By using IR style processing using inverted list and structure index.
d. By using a Relational Database Management System.
e. Techniques which utilizes efficient data structures like B^+ tree, hash table etc

A. Techniques using Inverted List

This technique makes use of the (Docid, Leftpos : Rightpos, levelnum) representation of XML elements and string values where Docid is unique document identifier, Leftpos and Rightpos can be obtained by counting word numbers from the beginning of the document till the start and end of the element respectively. For string values Leftpos and Right pos are same. This positional representation of XML document can correctly identify the binary structural relationship between the nodes of the tree. A tree node n2 which is represented as (D2, s2:e2, l2) is descendant of a tree node n1 represented as (D1,s1:e1,l1) iff D2=D1, s1<s2 and e2<e1. Similarly the node n2 will be child of n1 iff D2=D1, s1<s2, e2<e1 and l2=l1+1. Similarly the parent child relationship can also be identified exactly. Based on this representation of XML elements and string values Al – Khalifa *et. al.* [4] implemented the tree-merge and stack-tree algorithm for binary structural matching and then joining these matches. These algorithms are I/O and CPU optimal but produce large intermediate results. Similarly N Bruno *et.al.* [5] implemented the path stack and the twig stack algorithm. These algorithm do not produce large intermediate result but they need to examine every node in the input list to check whether or not it is part of an answer to the query(Path or twig) pattern i.e. it increases the search space and hence the response time.

B. Techniques using Structure Index and Inverted List

Raghav Kaushik et.al. [6] introduced the concept of structure index. Structure index is a compact summarization of the tree representation of XML document. It maintains only unique paths by eliminating all redundant paths in the tree. This structure index provides index ids for the index nodes which is used to augment the inverted list. The inverted list which makes use of the index id takes the form <Docid,start,end,level,indexid> for element node and <Docid,start,level,indexid> for text nodes. Here, docid refers to a unique document identifier and level is the depth of the node in the tree. The start and end numbers need to satisfy the following properties:

1. For each element node n, n.start < n.end.

2. If (element) node n1 is an ancestor of element node n2, then n1.start < n2.start < n2.end < n1.end.

3. If (element) node n1 is an ancestor of text node n2, then n1.start < n2.start < n1.end.

4. If element nodes n1 and n2 are siblings and ord(n1) < ord(n2), then n1.end < n2.start.A similar property holds when one or both of n1 and n2 are text nodes.

Using these index ids the search space can be reduced significantly. This is because only those inverted list entries that are part of the final result take part in the join process; other entries are skipped by just comparing its index ids. But still the search space is large since at least index id for every inverted list entry is to be checked.

C. Using Relational Database Management System

An XML document can be queried by the techniques already present in RDBMS. For example the different join algorithms like merge; hash, index nested loop join algorithm etc can be used. An XML document is decomposed in to relation consisting of four columns (Headid, schemapath, leafvalue, idlist) where headid is the id of the node from where the path starts, schemapath is the list of nodes on the sub path, leaf value is the text for an element or attribute represented in the form of string. Leaf value is present only if the path reaches the leaf. Otherwise a null is stored for the leaf value. On this relation two types of indexes are constructed the root path index and the data path index. The root path index can answer any root to leaf query efficiently where as the data path index can answer the queries with // axes. This technique requires that an XML document be first decomposed into a relation and an XML path query be translated into SQL query. If the elements are deeply nested then it will populate the table with large number of null values. i.e. it increase the index size [7][8].

1. Path and Content Index

This method makes use of Dewey id as ids for nodes and creates a separate structure index and value index. The id of a particular node contains the id of its parent as its prefix and some local id. Thus if the id of the parent is 1.2 then ids of its children will be 1.2.1, 1.2.2, 1.2.3 …….1.2.n. This makes it easier to determine the structural relationship such as the parent child and ancestor-descendant relationship between the nodes. For example the parent of the node 1.2.1 will be the node 1.2. In the similar fashion ancestor-descendant relationship between the two nodes can also be determined. Moreover the generation of a node can be determined by counting the number of sub ids in the id of a node in question. For example the node 1.1.2.1 is at fourth level (or at depth of four) from the root node. Path and content index employs a hash table which contains an entry for every element tag. For every distinct entry in the hash table there is a B^+ tree to store the Dewey id. The leaf node of a B^+ tree contains a set of related nodes. The node contains the Dewey id to identify the structural relationship and a pointer which points to the nodes contents. Thus the contents of arbitrary node can be accessed directly. Since the Dewey id for a node contains the id of its parent as its prefix the id of the node becomes very long as the depth of the tree increases. Hence it requires longer processing time to determine the binary structural relationship [9].

2. Structural Summary Index

The technique specified in [10] creates a compact structural summary index for XML data. In contrast to the Inverted list, structural summary index classifies identifiers of data nodes with same label path into a group. To support queries containing the // axis (i.e. ancestor – descendant or partially specified queries) efficiently the index stores steps of label path as a key in reverse order. For example the path bib.book.title is stored as title.book.bib. Any query beginning with // can be processed efficiently by prefix matching and this operation of prefix matching can be performed in a single index lookup on a B^+ tree. For example the queries like //title can be answered efficiently in single index lookup by using the structural summary index. For queries with non initial // the structural summary index requires longer time.

3. Compact Redundancy Free XML Storage

Most of the earlier techniques for Indexing of XML made use of tree representation of XML document which led to increase in space requirement. The technique specified in[11] proposes a combination of hierarchical and relational structure to store XML data. Thus it uses a non tree based structure. The RFX (Redundancy Free XML) storage is a multilayered architecture where the elements and data are stored as separate layers which facilitates fast navigation and retrieval of data.

It stores the XML document in three layers the topology layer, the tag layer, and the data layer. The XML files are first analyzed for their relationships in the database i.e. containment, intra or inter document relationship and then the XML files are stored conforming to the structural norms of the RFX system.

4. Structural Query Optimization in Native XML Database

The technique specified in [12] proposes a novel labeling scheme consisting of <self-level:parent> to support different binary structural relationship that exist in an ordered labeled tree representation of XML documents.

This technique makes use of the decomposition-matching merging approach to decompose the twig into a set of path queries. It utilizes the hybrid query optimization technique INLAB(combination of INdexing and LABeling technique)which consist of create INLAB encoding and TwigINLAB algorithms. The index structure of INLAB s finds efficiently all elements that belong to the same parent or ancestor. The proposed labeling scheme quickly determines the parent-child, ancestor-descendant, sibling and ordered query relationship between elements in the native XML database. This technique reduces the number of inspections performed on irrelevant results during the merging process and hence improves the response time.

5. Tbitmap(Tag bitmap) technique for Indexing of XML Document

The technique specified in [13] utilizes bitmap (sequence of bits of zeroes and ones) for representation of tags of XML elements. The bitmaps are assigned in such a way that it uniquely identifies an XML element and also correctly identifies the parent-child and ancestor-descendant relationship between the elements. A sample XML document using the Tbitmap is shown below in Figure 1:

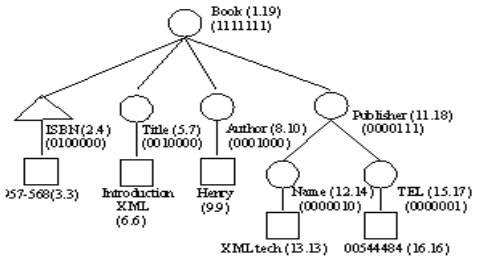

Fig. 1: A Sample XML Document with Tbitmaps

An XML document is preprocessed by performing following tasks:

1. A code book is created which assigns unique bit map to every structure element. It is created by performing bitwise OR operation to get the signature of all of its children.
2. A containing code is assigned to every element. This containing code is obtained by performing a depth first search on the XML document.
3. Two indexes namely the tag index and the value index are created on the XML document. The tag index identifies the structure. It is built using the B^+ tree. The containing code for a node is used as search key in the tag index. The value index contains the containing code and a pointer to the nodes contents. The tag index and the value index are shown in Figure 2.

If the number of nodes increases then the number of bits in the bitmap for structure node becomes very large. Processing of such bitmaps to determine the structural relationship takes longer time and hence results in longer response time.

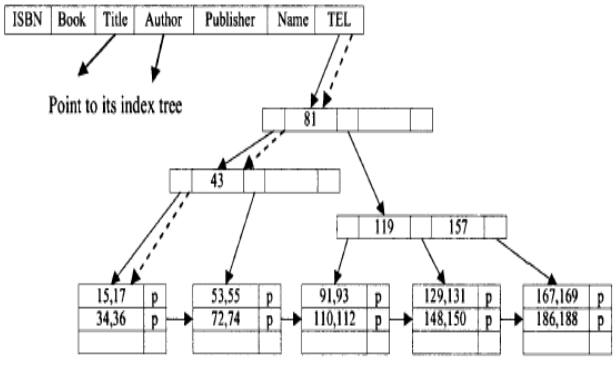

Fig. 2: A tag index tree

6. Hint and Run

Iona Stania *et.al* [14] proposed and implemented an innovative idea of using Hint to guide the query path. By using Hint value of a particular query node the search space can be reduced effectively. A hint h(l,c,t) at a node l returns a positive value if the element tag t is present in the sub tree rooted at node c. Node l is the parent of node c. Thus a negative value of hint indicates that the sub tree rooted at c can be skipped from traversing since it will not contain the element t. The efficiency of hints depends on two factors. The first one is the position or location of the node at which the hint value is calculated. The second factor is the number of elements for which the hint value is calculated. If the hint value is calculated at every possible location it will decrease the number of nodes to be traversed. Similarly if the hint value is calculated for every node at every possible location then it will access only the nodes which are present on the query path. In h(l,c,t) l,c,t are ids of nodes which are all integers and the hint value is also an integer. If it is assumed that integer requires two bytes then a hint at a particular node for a particular tag requires eight bytes of storage. The concept of hint can be used as standalone technique or as an augment to the existing query algorithm. A hint value is calculated only for element tag. It is not calculated for attribute or contents of an element. Thus such a hint value will not be useful for queries which are selective i.e. the queries which involves predicates.

7. XReal (Technique for XML Keyword Search)

This may be the first ever known technique which performs the keyword search on XML documents efficiently based on Information Retrieval style. It effectively handles the three issues in XML keyword search:

1). Effectively identifies the type of target nodes that a query keyword intends to search for. Such a node is termed as search for node.

2). Effectively infers the type of condition nodes that a keyword query intends to search for. Such condition nodes are termed as search via nodes.

3). Effectively ranks each query result in consideration of the above two issues.

This technique provides novel formulae to identify the search for nodes and search via nodes of a query node and presents a XML TF*IDF ranking strategy to rank the individual matches of all possible search intentions. This technique outperforms all the earlier known keyword search techniques which utilize the SLCA (smallest lowest common ancestor) for

keyword searching and ranking the search results [15].

8. XCDSearch(XML Context Driven Search Engine)

The technique specified in [16] answers XML keyword based queries as well as loosely structured queries using a stack based sort merge algorithm. Keyword based search query is user friendly since it does not require the knowledge of any query language or the structure of the underlying data. Loosely structured query is a query in which the user knows only the labels of elements containing the keywords but does not require the user to be aware of the exact structure of the underlying data.

Previous keyword search techniques focused on building relationship between data elements based solely on their labels and proximity to one another while overlooking the context of the element which many times led to erroneous results. This technique treats the parent and all its children as a set i.e. a single unified entity and then uses a stack based sort-merge algorithm to perform a context driven search to determine the relationship between different unified entities

All the above technique can be compared on the basis of various factors such as intermediate results size, index size, response time, I/O required, flexibility etc listed in the following Table 1.

Table: Comparative study of existing indexing techniques

		Performance factors				
Sr.No	Indexing Techniques	Intermediate Result size	Index Size	I/O required	Response Time	Flexibility
1	Tree Merge	Large	Large	Large	More	High
2	Stack Tree	Large	Large	Less	Moderate	High
3	Path Stack	Less	Large	Less	Less	High
4	Twig Stack	Less	Large	Less	Less	High
5	Inverted list Using structure index	Very Less	Large	Less	Less	High
6	SSI	Very Less	Small	Less	Very Less	Efficiency reduces when a query contains non initial //
7	T bitmap	Very Less	Small	Less	Less	Not suitable for Deeply nested tree
8	Pc Index	Very Less	Moderate	Less	Less	Not suitable for Deeply nested tree
9	Hint & Run	Very Less	Large	Less	Less	Less

III. CONCLUSION

In this paper we have reviewed several indexing and querying techniques for XML. From this in depth review we conclude that the response time for XML query depends upon several factors such as indexing /storing technique, intermediate result size, index size, amount of I/O required etc. The most important factor being the indexing technique used to store and query XML data. Efficient and innovative data structure can be to create powerful index structures. This index structure coupled with efficient query algorithm can answer highly flexible XML queries with very less response time

IV. REFERENCES

[1]. Wolfgan Meir," Exist: An Open Source Native XML Database," http:// exist.sourceforge.net/2008.

[2]. "W3c consortium XML path language(Xpath) 2.0," http//www.w3.org/tr/xpath20/2007.

[3]. "W3c consortium Xquery1.0 : An XML query language," http//www.w3.org/tr/xquery/2007.

[4]. S. Al. Khalifa, H. V. Jagdish, N Koudas, J. M. Patel, D Srivastava and Y Wu," Structural Joins: A Primitive for Efficient XML Query Pattern Matching," In Proc. of the *18th International Conference on Data Engineering (ICDE*), San Jose, CA, pp. 141-152, February 26-March 1, 2002

[5]. N. Bruno, N. Koudas, and D. Srivastava, "Holistic Twig Joins: Optimal XML Pattern Matching", In Proc. Of *21st ACM SIGMOD Int'l Conference on Management of Data* (SIGMOD'02), pp. 310–321, 2002.

[6]. Raghav Kaushik, Rajasekar Krishnamurthy, Jeffery F. Naughton, Raghu Ramkrishnan," On the Integration of Structure Index and Inverted List," In Proc. of the *204 ACM SIGMOD international conference on Management of data*, Paris, France, pp.779-790, June 13-18 2004

[7]. Zhuyan Chan et. Al," Index Structures for Matching XML Twigs using Relational Query Processor," In Proceeding of *Data engineering workshop ICDEW* ,5-8 April 2005.

[8]. Igor Totarinov, Stratis D Vigals, Kevin Beyer et.al.," Storing and Querying Ordered XML using a Relational Database System," In Proc. Of *ACM SIGMOD Int'l Conference on Management of Data,* Madison Wisconsin USA, *pp. 204-215, 2002*.

[9]. Li Ying, MaJun Sun Yun, "Applying Dewey Encoding to Construct XML Index for Path and Keyword Query," In Proc. Of *First International Workshop on Database Technology and application 09*,pp553-556, 25-26 April 2009.

[10]. Xiaojie Yuan XinWang, Chenying Wang," Efficient Xpath Evaluation Using Structural Summary Index," In proceedings of *International Conference on Computer Science and Software Engg*, 2008.

[11]. Radha Senthilkumar, Priyaa Varshinee and A. Kannan(2009, June). Designing and Querying a Compact Redundancy Free XML Storage. *The Open Information System Journal. 3*, pp. 98-107.

[12]. Su-Cheng Haw and Chien-Sing Lee(2007). Structural Query Optimization in Native XML Database: A Hybrid Approach. *Journal of Applied Sciences. 7(20)*, pp. 2934-2946.

[13]. Yin Fu Huang and Shin-Hang Wang," An efficient XML Processing based on combining T bitmap and Index Techniques," IEEE *Symposium on Computers and Communication ISCC* 2008, Marrakech, Morocco, July 6-9 2008.

[14]. Iona Stanoi, Christan A. Lang, Sriram Padmanbham," Hint and Run: Accelerating Xpath Queries," In Proceedings of the *9 th International Database Engineering and Application Symposium IDEAS* ,2005.

[15]. Zhifeng Bao, Jiaheng Lu, Tok Wang Ling(2010, August). Towards an Effective XML Keyword Search. *IEEE Transaction on Knowledge and Data Engineering. 22(8)*, pp. 1077-1092.

[16]. Kamal Taha, Ramez Elmasri(2010, December). XCDsearch: An XML Context Driven Search Engine. *IEEE Transaction on Knowledge and Data Engineering. 22(12)*, pp. 1781-1796.