International Journal of Future Generation Communication and Networking
Vol. 13, No. 4, (2020), pp. 364–374

A Comparative Study of Keyword Searching on XML Trees Using Compact Tree Indexing and XR Tree Indexing

JK Swapna, Dr G. Vijaya Lakshmi

[1]JK Swapna, Ph.D Research Scholar, Department of Computer Science, Vikrama Simhapuri University, Kakutur, Nellore-AP, swapnajk2001@gmail.com

[2]Dr G. Vijayalakshmi, Assistant Professor, Department of Computer Science, Vikrama Simhapuri University, Kakutur, Nellore-AP, vijaya_suma17@yahoo.co.in

Abstract

XML documents have been rapidly used to store heterogeneous data types which could be either structured or semi structured. Large organizations prefer to store their data in the format of XML as it offers many benefits compared to HTML. Due to the rise in the usage of XML databases, the need to locate the data and maintenance of it has been in demand. Keyword Searching is one of the techniques where the user can locate a data item without knowing the query semantics or the structure of the data stored. To assist in easy retrieval of specific information, indexing the data efficiently helps a lot. This paper primarily focused on using a compact tree structure to index the XML data and developed an algorithm on top of it which assists in quick query processing and minimized index traversal time. We also studied a labelled tree known as XML region tree and presented the comparative results to show the efficacy of our indexing structure.

Keywords: XML, Keyword Searching, Compact Tree, XML Region Tree, Indexing

1. Introduction

XML databases are object based databases which helps us to capture data of an organization which is in hierarchical structures. It is a tagging markup language a little different from HyperText MarkUp Language. It can be used to store data or use it to link organizational data. Once data is stored in the XML database, to retrieve it a user should learn schema specific XML languages such as XPath, XQuery etc which is cumbersome. To ease the user, keyword searching came into

ISSN: 2233-7857 IJFGCN

existence which helps any naive user who knows searching using a simple search tool. The area of Keyword Search is broadly divided into two phases: Indexing and Ranking.

There have been many traditional indexing techniques which used to consider an entire document as a set of words and used to find the keywords belonging to the document or not. These conventional indexing won't work for databases which are semi structured and where data is related with more that one document. Building XML Indexes in contrast to traditional Indexes is difficult as most of the XML data is represented in tree structures. Research has been done on how to build XML indexes[1] and its various operations such as Index Updation and also algorithms to decrease the query processing time in keyword searching using these indexes.

This paper focuses mainly on various structural indices which could assist faster keyword searching and reduced index traversal time to locate the keywords. Section 2 briefly presented the existing methods to address the challenges of indexes. Section 3 discusses using Compact Ttree (CTree) [2] indexing structure which addresses the existing challenges. Section 4 describes the XR Tree[3] Indexing which also uses a similar kind of indexing approach. Section 5 presents the comparison results of the proposed system with XR Tree Indexing [3] and conclusion remarks.

2. LITERATURE REVIEW

Any indexing structure to be most efficient must be able to retrieve the sub trees of XML documents which represent related keywords. In one way XML index must be able to process the trees top-down or bottom-up by traversing the tree nodes. The results could be simple paths or complex paths which are known as twigs. Every node in the XML tree is assigned a Label which gives the node

information such as its parent, child, siblings, in which level it is located etc. These labels play a very important role in accelerating query processing.

Let's take an example query article[editor="ESHWAR"] and see how it is evaluated in Path traversal systems. In top-down processing, the query is processed by looking at all downward paths starting from any article element which has an immediate editor element. It then traverses downward to find the article edited by "ESHWAR". Next, it looks for all the nodes with labels in the document to determine all possible paths. To process a search query consisting of "title" and "journal", it needs to traverse all the paths from top to leaf nodes which consists of a journal sub element. In general there would be more than one match, so it needs to find its way back to top to search for the next child node which contains the "journal" element.

This is a time taking, in-efficient method to search the matching keywords in the XML trees. So, we move on efficient indexing techniques which overcome exhaustive path traversals and are also effective in processing parent-child related queries.An index technique would be proved as efficient if it helps in deducing the relationship between the nodes present in different levels which are encoded in the index files [4]. It should also help in knowing the attributes of the document such as number of levels, depth of the tree, siblings, ancestors, descendants. The index file must be able to manage the load of the hierarchical XML document.

Gou et al [5] had categorized two types of indices based on the traversals. a) XML Path Indices are used for processing the simple path queries. b) XML Twig indices were used for processing twig queries. In the processing of twig query, a subsequent joining of simple paths is required. In comparison with the node indexing scheme, the path indexing scheme requires less number of joins in query processing, thereby improving the performance.

In contrast to above, Vakali et al[1, 6] summarized different indices on type of data stored and the structure of the XML document. They defined a structural summary index structure which preserves all the paths from root nodes to leaf nodes. This eliminated book keeping information about the hierarchical structure and still maintained the relationships between two nodes. The main advantage of these indices are they are very effective in processing path queries, but not suitable for twig queries. Similar to Gou et al[5], path indices were used to store the label paths whereas Node indices were defined to store node names and joins were needed to reconstruct the structure. The latter most sequence based indices stored data related to both the documents and queries in a sequential manner to match the keyword queries by sequence matching.

Path Index Structures mainly focus on root to leaf paths and don't bother about the content. They used to take the help of a supplementary value index for the content. Because of this, the query processing cost involves not only joins but also recursive lookups in the index to match the keywords. An index structure which stores both the path and content was devised by Cooper et al. [7]. However, it cannot support ancestor-dependent queries efficiently. They cannot efficiently process the partial match or content-based queries.

Node Index Structures [8, 9] are based on the labeling schemes which can be containment based or prefix-based. It holds the value that depicts the node's position in the XML document. These indices support both parent-child and ancestor-dependent relationships and it needs only two comparisons to infer. But a challenging task is to have cost effective index updes.

The XML tree and the query is converted to a sequence in Sequence Structural Index structures and then it uses a subsequence matching algorithm to evaluate the query results. Structural-encoding sequences are defined by Haixun et al [10] to store the XML data and the search query. The SES consists of two components.

The first component is the element tag and the second is the path of its parent starting from the root node. The encoding starts by scanning the data tree in a top-down approach. The length of the sequence will increase as the path of the XML data tree gets longer. Label length increases as the tree depth increases. For longer trees the index updation is a challenging task.

Praveen et.al [11] overcomes the limitation of Haixun [12] which eliminated the need to evaluate each and every node by removing the duplicate nodes. In contrast of top to down sequencing they used bottom up sequencing to minimize the cost of query evaluation.

Hence there is a dire need to devise an index structure which can help in overcoming the limitations.

1. It should be able to preserve both content and structural properties on XML documents.

2. To infer all the relationships between any two nodes.

3.Index file size should be less and reduced index file updates

4. Speed up query processing.

In the proposed study we use an indexing structure which addresses all the above issues and have the advantages of Path based and Node based indexing. Compact Tree Index (Ctree) [2, 13] is used to index the XMl documents which stores not only the path summaries which preserve the Parent-Child relationships but also detailed summary of ancestor to dependent relationships at element level. It also speeds up the query processing as it prunes out a large number of irrelevant nodes and matches the context using an inverted index.

3. Proposed Index Method:

Ctree[12] is a form of binary tree with two levels which is used to summarize the entire XML document in a compact manner. Each node in CTree has two pointers, where the group pointer points to the nodes of similar child nodes having the same

parent node and the element pointer stores data about its children nodes as well as respective parent nodes. Building of group pointers is usually done by identifying all the nodes with similar hierarchical sub structures who have a common parent node and storing them as a group. In the next level, each node holds the pointers sequentially to its list of parents and children. With reference to the Ctree properties, we found that if we built an efficient indexing algorithm which indexes the Groups and Elements (represented in ctree), it would decrease the number of comparisons to search a keyword in the XML tree. Due to less number of comparisons to locate a keyword, it gradually decreases the query processing time. So we devised a novel indexing algorithm using ctree which could be used for XML keyword Search. We also used the level at which the keywords appeared to calculate the proximity.

The proposed work is divided into three phases as shown as in the design in Figure1.

Phase 1 - Ctree Builder and Ctree Index Generator : Using Ctree based indexing to index the XML documents. This requires XML documents to be parsed and storing them in a relational database in the form of tables. And building an index on this tabular data.

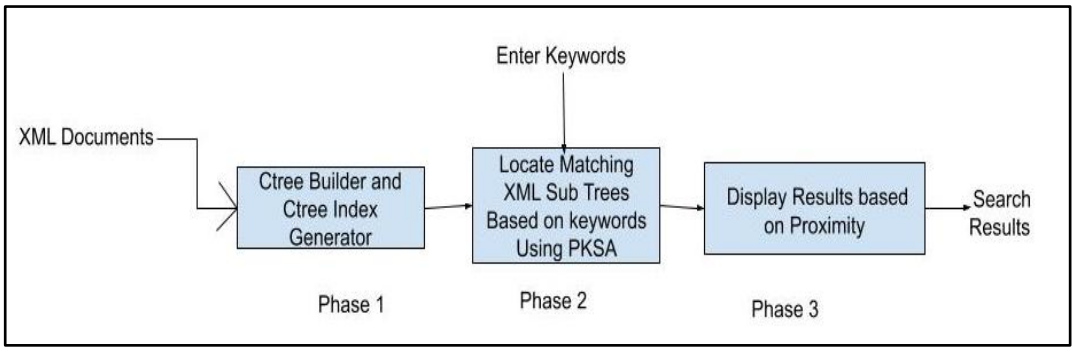

Figure 1 : Proposed Design

Phase 1 - Ctree Builder and Ctree Index Generator : Using Ctree based indexing to index the XML documents. This requires XML documents to be parsed and storing them in a relational database in the form of tables. And building an index on this tabular data.

Phase 2 - Locate Matching XML Subtrees based on Keywords using Proximity keyword Search algorithm (PKSA) : The second major step is to efficiently use the Ctree index to compute the XML subtrees which contain all the keywords entered by the user.

Phase 3 - Display Results based on Proximity : The final step is displaying the XML subtrees by ranking them based on edge distance from the Lowest Common Ancestor of the elements which contain the keywords.

The generated Ctree index locates the keyword in the XML elements and returns a list of XML tags based on the element ids and group ids of where the keyword in matched in the node. Then a minimum connecting tree with the lowest common ancestor of the given keywords are computed using our algorithm .

1. Find the group ids and element id's of the given keywords from the index table and store it in two lists.

2. If the group id's of all the keywords are the same Check their element id's are equal.

(a) If they are equal Display the element id along with the given keywords

(b) If they are not equal Compute the LCA of the keywords by retrieving their parent element ids and group ids.

else

(a) Retrieve the depth of each keyword. Let p and q be the keywords which are at maximum depth and minimum depth respectively.

(b) Recursively reach the ancestor of every keyword which is at level(q) from the keywords which have depth less than equal to p.

(c) Compute the LCA of the ancestors.

3. Rank the results based upon the distance between the keywords.

In addition to computing the minimum connecting trees containing the keywords , a metric known as score is also computed for every XML document. Let's assume the user has submitted n keywords. If a XML document contains all n keywords, its score is dened as 100. With n keywords we can nd n! combinations. If a XML document contains less than n number of keywords say p, its score is dened as 100 - ((p/n!) * 100). For example, with 3 keywords, there are 6 possible combinations. Score of a XML document which contains all 3 keywords is 100 percent. Score for an XML document which contains 2 keywords is 100 - ((2/6) * 100).

4. XML Region Tree Index Structure :

XR (XML Region)[24] Tree are proposed on the basis of a numbering scheme to index XML documents. It is dynamic in nature which gives efficient index updates with no time. Each node in this tree is represented by a region similar to an entity/object with beginning and finishing node numbers [25, 22, 8] based on its location in the entire xml document such that for any two nodes n1 and n2, either entire n2 is before or after n1 and also either n1 contains n2 or n2 contains n1. Depth First Traversal is used to enumerate the nodes in the regions. If we clearly understand the B+ Trees, we can infer that an XR tree is a B+tree in simplicity. Each region stores complex entries with extra pointers pointing to its before and after regions. They use a concept known as stabs where each stab entry is entered into an index entry.

5. Comparison of XR Tree Index structures with Compact Tree Index Structures :

We used a sample data set from the DBLP[26] database to compare the efficiency of Compact Tree Index Structures with XR Tree Index Structures. We simulated a B+tree index structure to evaluate the parent-child and ancestor dependent relationships and compared the performance of Ctree Index with XR Tree structures with respect to Index file Size and some path based queries. We compared the effectiveness of compact tree index entries with respect to building

indexes with XR Tree[25]. We experimented with Xpath queries where the keywords are at different levels and also queries which results in twig query results . We found that query processing time for simple path queries takes almost the same time for both the indexes. For complex queries which are at different levels, compact tree index fairs much better than XR Trees. For nodes which needs traversal of more than one document Ctree efficiently traverses within a short span of time as well as query processing time is reduced.

Table 1: Processing times of Ctree and XR Tree Indexed Algorithms

Parameters	CTree Indices		XR Trees	
XPath Queries Experimented	QPT	ITT	QPT	ITT
/article/title/author	65	65	68	85
//year/author	345	347	396	412
//article/title/author/p[contains(., "Harry")]	86	86	94	98
//article[contains(./fm/abs/p, "Harry")]	4567	4560	5672	5767
//dblp/session[1]/paper[contains(., "TOM")]	769	780	823	896

QPT = Query Processing time in milli secs, ITT=Index Traversal time in milli secs

6. Conclusion:

In this paper we tried to compare two different XML index structures Ctree and XR tree. Both are good in preserving relationships between different nodes. XR tree is mainly used to construct an index for strictly nested XML data whereas Ctree could be used for any kind of complex data trees. Space overhead as well as index updation is less for Ctree indices compared to documents indexed by XR trees.

References

[1] Barbara Catania, Anna Maddalena, Athena Vakali " XML Document Indexes", Proc. IEEE Internet Computing, SEPTEMBER - OCTOBER 2005, pp 64 -71.

[2] Qinghua Zou, Shaorong Liu, Welsley W.Chu, “Ctree: A Compact Tree for Indexing XML Data”, in WIDM 2004.

[3] Haifeng Jiang, Hongjun Lu, Wei Wang and B. C. Ooi, "XR-tree: indexing XML data for efficient structural joins," Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405), Bangalore, India, 2003, pp. 253-264, doi: 10.1109/ICDE.2003.1260797.

[4] I. Tatarinov et al., “Storing and Querying Ordered XML Using a Relational Database System,” Proc. Int’l Conf. Management of Data (ACM Sigmod), ACM Press, 2002, pp. 204–215.

[5] Q. Li and B. Moon, “Indexing and Querying XML Data for Regular Path Expressions,” Proc. Int’l Conf. Very Large Databases (VLDB 01), Morgan Kaufmann, 2001, pp. 361–370

[6] YP.E. O’Neil et al., “Ordpaths: Insert-Friendly XML Node Labels,” Proc. Int’l Conf. Management of Data (ACM Sigmod), ACM Press, 2004, pp. 903–908.

[7] R. Goldman and J. Widom, “DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases,” Proc. Int’l Conf. Very Large Databases (VLDB 01), Morgan Kaufmann, 1997, pp. 436–445.

[8] W. Wang et al., “Efficient Processing of XML Path Queries Using the Disk-Based F&B Index,” to appear, Proc. Int’l Conf. Very Large Databases (VLDB), Morgan Kaufmann, 2005.

[9] T. Milo and D. Suciu, “Index Structures for Path Expressions,” Proc. Int’l Conf. Database Theory (ICDT 99), LNCS 1540, Springer-Verlag, 1999, pp. 277–295.

[10] C.W. Chung et al., “APEX: An Adaptive Path Index for XML Data,” Proc. Int’l Conf. Management of Data (ACM Sigmod), ACM Press, 2002, pp. 121–132.

[11] Praveen rao, Bongki Moon, “PRIX: Indexing and querying XML using prufer sequences”, Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA.

[12] Haixun Wang, Sanghyun Park, Wei Fan, PhilipS.Yu, "ViST: A DynamicIndex MethodforQueryingXMLDataby TreeStructures" SIGMOD 2003,June9-12,2003,SanDiego,CA.Copyright2003ACM1-58113-634-X/03/06.

[13] S. Al-Khalifa et al., “Structural Joins: A Primitive for Efficient XML Query Pattern Matching,” Proc. Int’l Conf. Data Eng. (ICDE 02), IEEE CS Press, 2002, pp. 141–152.

[14] Lin Guo, Feng Shao, Chavdar Botev, Jayavel Shanmugasundaram, “XRank: Ranked keyword Search over XML Documents," in SIGMOD, 2003.

[15] Sara Cohen, Jonathan Mamou, Yaron Kanza, Yehoshua Sagiv, “XSEarch: A Semantic Search Engine for XML," in Proceedings of the 29th VLDB Conference, 2003.

[16] L. M. Amini and M. Keyvanpour, "On user-centric XML keyword search," 2018 4th International Conference on Web Research (ICWR), Tehran, 2018, pp. 51-57, doi: 10.1109/ICWR.2018.8387237.

[17] Roko Abubakar , Shyamala Doraisamy , Bello Nakone, "Effective Predicate Identification Algorithm for XML Retrieval", 2018 Fourth International Conference on Information Retrieval and Knowledge Management (CAMP)

[18] Yushan Ye , Kai Xie , Tong Li , Nannan He, "Result ranking of XML keyword query over XML document" , 2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)

[19] T. Chen, J. Lu, and T.W. Ling, “On Boosting Holism in XML Twig Pattern Matching Using Structural Indexing Techniques,” Proc. Int’l Conf. Management of Data (ACM Sigmod), ACM Press, 2005, pp. 455–466.

[20] H. Jiang et al., “XR-Tree: Indexing XML Data for Efficient Structural Joins,” Proc. Int’l Conf. Data Eng. (ICDE 02), IEEE CS Press, 2002, pp. 253–263.

[21] A. Silberstein et al., “BOXes: Efficient Maintenance of Order-Based Labeling for Dynamic XML Data,” Proc. Int’l Conf. Data Eng. (ICDE), IEEE CS Press, 2005, pp. 285–296.

[22] B. Catania et al., “Lazy XML Updates: Laziness as a Virtue of Update and Structural Join Efficiency,” Proc. Int’l Conf. Management of Data (ACM Sigmod), ACM Press, 2005, pp. 515–526.

[23] H. Wang et al., “ViST: A Dynamic Index Method for Quering XML Data by Tree Structures,” Proc. Int’l Conf. Management of Data (ACM Sigmod), ACM Press, 2003, pp. 110–121.

[24] P.R. Raw and B. Moon, “PRIX: Indexing and Querying XML Using Prüfer Sequences,” Proc. Int’l Conf. Data Eng. (ICDE), IEEE CS Press, 2004, pp. 288–300.

[25] H. Wang and X. Meng, “On the Sequencing of Tree Structures for XML Indexing,” Proc. Int’l Conf. Data Eng. (ICDE), IEEE CS Press, 2005, pp. 372–383.

[26] Michael Ley. DBLP database web site. http://www.informatik.uni-trier.de/ley/db.