ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 10, October 2013

A NOVEL WAY OF INFORMATION RETRIEVAL FROM XML USING EFFICIENT HOLISTIC ALGORITHMS

Ravi Kiran Yandrapragada[1], Poreddy Dayakar[2]

M.Tech, Computer Science Engineering, MLRIT, Hyderabad, Andhra Pradesh, India[1]

Assistant Professor, Department of CSE, MLRIT, Hyderabad, Andhra Pradesh, India[2]

Abstract: XML has become a de facto standard in industry as a data carrier across multiple technologies. XML parsing is one of the key challenging technologies. The process of identifying all the occurrences of a twig pattern specified by a given predicate on multiple data elements in an XML document is a basic fundamental operation for efficient evaluation of XML queries. Hence the concept of indexing and querying of XML documents efficiently has been a major research issues in the XML community. Much research has been done for providing an efficient way to evaluate twig patterns in an XML database. As a result, many holistic join algorithms have been developed, most of which are derivatives of the well-known Twig Stack algorithm.

In this paper, we came up with a novel way of algorithm which helps in retrieving information from XML using various functions, which may include like Boolean function, comment node function, count function, wildcards and order restriction, negation functions.

1. INTRODUCTION

Since the evolution of software technologies the main way of generating and exchange of information in a common unique fashion is through XML and hence, there is an increasing need for efficient processing of queries for information on XML data. An XML query pattern can be represented as a rooted labeled structured tree often referred to as twig [1]. For example, consider the following XQuery path expression:

book [title = 'XSL'] //author[. = 'Lincoln']

The above expression matches the author elements that (i) have as content the string value "Lincoln", and (ii) are descendants of book elements that have a child title element whose value "XSL". This XQuery path query expression can be equated to a node-labeled tree pattern with elements and string values as node labels. Such a complex structured query tree pattern can be naturally decomposed to a set of basic parent-child and ancestor-descendant relationships between pairs of nodes. For example, the basic structural relationships corresponding to the above query are the ancestor descendant relationship (book, author) and the parent-child relationships (book, title), (title, XSL) and (author, Lincoln). The query pattern can then be matched by (i) matching each of the binary structural relationships against the XML database, and (ii) "stitching" together these basic matches. Efficiency of XML tree patterns matching has been considered widely as one of the core operation in XML query processing. In recent years, many methods ([9], [14], [3], [11], [12], [15]) have been proposed to retrieve information from XML tree queries efficiently and effectively. To consider, in particular, Khalifa [1] proposed a stack-based algorithm which helps in matching of binary structural relationship including parent-child (P-C) relationship and also the ancestor-descendant (A-D) relationships. The limitation of Khalifa approach is that the size of useless intermediate results may become to extremely very large, even if the final results produced is very small. Bruno proposed a novel way of holistic twig join algorithm and named it as TwigStack, which processes the tree pattern holistically without decomposing it into several tiny binary relationships. The TwigStack algorithm guarantees that there are less/no "useless" intermediate results for queries with only ancestor-descendant (A-D) relationships. In other words, the algorithm can be TwigStack is suitable for tree pattern queries with only A-D edges [8]. We have proposed a holistic algorithm called TreeQueueMatch which helps in bringing out results from XML using various functions like negation function, Boolean function (TQ1),comment node function (TQ2), count function (TQ3), wild card function (TQ4), order restriction function (TQ5), negation function (TQ6) as shown in the Figure 1.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 10, October 2013

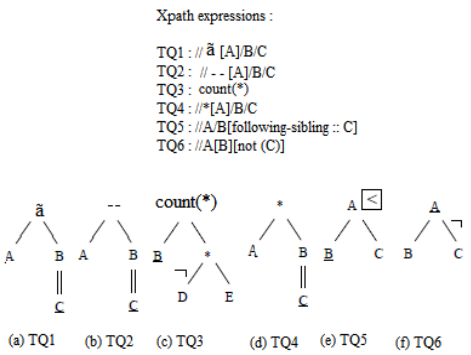

Fig 1: Various functions of TreeQueueMatch

Many new and recent works examine how to enlarge the optimality of query class of holistic algorithms [14], to speed up the performance using indexes [11], [5], to devise a new data streaming strategies [6], and to propose an efficient and dynamic labeling schemes [16]. These algorithms have proven highly to be promising and make their way groundbreaking in XML query processing applications, both in academic and industrial settings [19].

2. RELATED WORKS

In the context of semi-structured XML databases, tree based query pattern is a very practical and is an important class of queries. Lore DBMS [9] and Timber [10] systems have considered various aspects of query processing on such XML data and queries. The XML data and various issues in their storage as well as query processing using relational database systems have recently been considered in [8], [17], [12], [19].

From a theoretical point of research about the optimality of XML tree pattern matching, Choi et al. [17] developed theorems to prove that it is very difficult to devise a holistic algorithm to guarantee the optimality for queries with any combination of P-C and A-D relationships. Shalem and others [14] researched the space complexity for the processing of XML twig queries. Their article showed that the upper bound of full-fledge queries with parent-child and ancestor-descendant edges are O(C), where C is the document size. In other words, their results have also theoretically proved that there exist no such algorithms to optimally process an arbitrary query. Our research in this article moves the frontier a bit forward by identifying a large subclass of XML which can be guaranteed to process optimally. The recent papers (e.g. [17], [12], [12], [6], [5]) have also closely related to ours. In paper [17], a new holistic algorithm, called OrderedTJ, has been proposed to process order-based XML tree query. In paper [16], [12], [10] an algorithm called TwigStackList is proposed to handle queries with negation function. In this article, we follow the line of holistic XML tree pattern processing and give a complete solution to efficiently process extended XML tree queries with wildcards, negative predicates and ordered/unordered restriction.

3. PROPOSED MODEL

In this article, we call an XML tree pattern with comment node function, count function, Boolean function, negation function, wildcards and/or order restriction as extended way of XML tree pattern. Figure 1, for example, shows six new XML tree design patterns. Query (a) includes a boolean function. Query (b) includes a comment function so that we can get comments in XML data that matches the given predicate. Query (c) includes a count function to count the total number of elements that matches a given query (d) includes a wildcard node "*", which can match any single node in an XML database. The Query (e) specifies and order restriction which can be placed on the given query while retrieval of information for the data. Query (f) includes a negative edge, denoted by '¬'.In XPath language [2], the semantic of negative edge can be presented with "not" boolean function. Query (e) has order restriction, which is equivalent to an XPath "//A/B [following-sibling::X]".

The ordering '<' symbol in a figure 1 shows that all the children under A are ordered. The semantics of the ordered-base tree design pattern is captured by mapping pattern nodes to the nodes of an XML data, such that the structural and ordered relationships are finally satisfied. Finally, Query (d) is more complicated, which contains wildcard. Previous XML tree design pattern matching algorithms does not fully exploit the use of "optimality" in holistic algorithms. The TwigStack [3] guarantees that there is no useless intermediate result for queries with only Ancestor-Dependent (A-D) relationships.

Therefore, The TwigStack algorithm is optimal for queries with only A-D edges. Another algorithm called TwigStackList [14] enlarges the optimal query class of TwigStack by including Parent-Child (P-C) relationships in non-branching edges. It has become a quite natural question that whether the optimal query class of TwigStackList can be further improved or not. And hence, the current open problems also includes on (1) how to identify a larger query which can processed optimally and (2) how to efficiently process a query which cannot be guaranteed to give optimal results. Note that earlier works in [13], [8], [21] have already showed that no algorithm is optimal for queries with any arbitrary combinations of A-D and P-C relationships.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 10, October 2013

```
Procedure TreeQueueMatch(docElement e)
Queue documentPath;
docElement currentElem;
1. BEGIN
2. WHILE documentPath not empty AND documentPath.top is not e's ancestor
3. currentElem = documentPath.pop();
4. FOR each query node E with matching label of currentElem
5. TreQueueMatch(currentElem, HS[E]);
6. documentPath.push(e);
7. END
```

Fig 2: TreeQueueMatch Algorithm

This article also explores the challenges and also shows the promise of a novel theoretical framework called "TreeQueueMatch" to identify a large optimal query class for posing extended XML tree queries. The TreeQueueMatch algorithm is specified in figure 2. Given a document element e which is visited in post-order, we will first check if e can be pushed into the corresponding hierarchical stack HS[E], or in other words, if all E's have child axes that can be satisfied. Once the document element e satisfies all the axes requirements for query node E, then we push e into the hierarchical stackHS[E]. Meanwhile, we need to maintain the hierarchical structure of the elements in the hierarchical stack HS[E]. To achieve this, we also merge the stack trees in HS[E] based on e and push e to the top of the merged stack). And when there are no existing stack tree that is the descendant of e, a new stack will be created to hold e. In general, given an extended XML tree pattern query which may include P-C, A-D relationships, comment function, order restriction, count function, negation function and boolean as well as wildcards functions, we consider the problem efficiently by matching the XML tree query. Our algorithm aims at identifying a large query which can be optimally processed. Similar to the previous papers on XML tree pattern matching (e.g. [3], [12], and [16]), in this article, we call a holistic algorithm called TreeQueueMatch which is "optimal" for a kind of query class, and it guarantees that any output intermediate results contribute to final answers.

4. PROTOTYPE IMPLEMENTATION

The prototype application is built to demonstrate as a proof of concept and also to offer a comprehensive evaluation of our new holistic algorithms also we have conducted experiments on both synthetic and real XML data. The synthetic dataset is generated randomly. The environment used to build the application includes a PC with 2 GB RAM, Intel I7 2.3GHz CPU processor running Windows 7 operating system. The platform used to develop the application Java 7.0 using NTFS file system as a simple storage engine and the development tool is done using Eclipse Galileo.

5. EXPERIMENTAL RESULTS

In this section, we have presented an experimental study of new TreeQueueMatch algorithm on the real-life and synthetic data sets. Our results have also demonstrated the effectiveness, in terms of accuracy as well as optimality, of the TreeQueueMatch holistic algorithms for large XML data sets. The benefits have been comparing to other four algorithms which were proposed much earlier called TwigStack[3], TJFast[16], OrderedTJ [17] and TwigStackList [12]. To experiment, we have considered a Small size of main memory in the first experiment; here we did not allow the output list in TreeQueueMatch algorithm to buffer any elements in the main memory, meaning that any element added to output list should be output to the secondary storage.

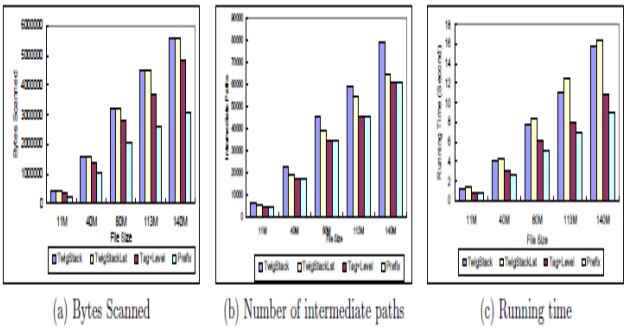

Fig 3: Experimental results of TreeQueueMatch

Then the requirement for main memory size is quite small. The purpose of this experiment is to simulate the application where the document is extremely large but the available main memory is relatively small. We made the experiments by using three different sizes of random documents. In particular, E1 has 210K nodes and E2 has 700K nodes and E3 has 1.2M nodes. Interestingly, Q6 is optimal for E1 and E2, but only slightly sub-optimal for E3. This can be explained that E3 is a larger document than E1 and E2 so that E3 manifests the sub-optimality which is hidden in E1 and E2. The output result of our experiment can be depicted in fig 3.The advantage is due to the fact that TreeQueueMatch guarantees that (almost) all of output elements belong to final results, which, in general, avoids the unnecessary I/O cost for outputting the useless intermediate results. The experimental results demonstrate that the number of output elements in TreeQueueMatch is always much less than that that of TwigStack and TJFast for different sizes of main memory. The reason due to that TwigStack and JFast buffer the intermediate results in the main memory and reduce the output of intermediate results. But the numbers of output elements in TreeQueueMatch

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 10, October 2013

remain the same, which always equals the final result size. From all tested queries, TreeQueueMatch has better performance than the previous algorithms. We contribute this improvement to the larger optimal query class TreeQueueMatch algorithm achieves. Since all the six queries contain negation, wildcards, comment, negative edge and order restriction, only our TreeQueueMatch can answer such complicated queries. The execution times of Q19 and Q25 are 16 and 12 seconds, respectively. Note that the above execution performance is achieved by using a relatively very small buffer size, we expect that our system can scale well for even gigabytes of XML data based on the current machine.

6. CONCLUSION

We have introduced a new holistic algorithm called as TreeQueueMatch and also introduced new functions to retrieve information using various functions. We have also identified a large optimal query classes for six different kinds of queries, that is ã, - -, count, //*, <,¬ respectively. Based on the obtained results, we have identified that this algorithm is best to achieve such an optimal query results. Finally, our extensive experiments demonstrate the advantage of our holistic algorithms on XML to verify the correctness of theoretical results.

REFERENCES

[1] S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, and D. Srivastava. Structural joins: A primitive for efficient XML query pattern matching. In Proc. of ICDE Conference, pages 141–152, 2002.
[2] A. Berglund, S. Boag, and D. Chamberlin. XML path language (XPath) 2.0. W3C Recommendation 23 January 2007 http://www.w3.org/TR/xpath20/.
[3] N. Bruno, D. Srivastava, and N. Koudas. Holistic twig joins: optimal XML pattern matching. In Proc. of SIGMOD Conference, pages 310–321, 2002.
[4] C. Y. Chan, W. Fan, and Y. Zeng. Taming xpath queries by minimizing wildcard steps. In Proceeding of VLDB, pages 156–167, 2004.
[5] S. Chen, H.-G. Li, J. Tatemura, W.-P. Hsiung, D. Agrawal, and K. S. Candan. Twig2stack: Bottom-up processing of generalized-tree-pattern queries over xml document. In Proc. of VLDB Conference, pages 19–30,2006.
[6] T. Chen, J. Lu, and T. W. Ling. On boosting holism in xml twig pattern matching using structural indexing techniques. In SIGMOD, pages 455–466, 2005.
[7] S. Chien, Z. Vagena, D. Zhang, V. J. Tsotras, and C. Zaniolo. Efficient structural joins on indexed XML documents. In Proc. of VLDB, pages 263–274, 2002.
[8] B. Choi, M. Mahoui, and D. Wood. On the optimality of the holistic twig join algorithms. In Proceeding of DEXA, pages 28–37, 2003.
[9] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In Proc. of VLDB, pages 436–445, 1997.
[10] H. V. Jagadish and S. AL-Khalifa. Timber: A native XML database. Technical report, University of Michigan, 2002.
[11] H. Jiang et al. Holistic twig joins on indexed XML documents. In Proc.of VLDB, pages 273–284, 2003.
[12] H. Jiang, H. Lu, and W. Wang. Efficient processing of XML twig queries with OR-predicates. In Proc. of SIGMOD Conference, pages 274–285,2004.
[13] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In Proc. of VLDB, pages 361–370, 2001.
[14] J. Lu, T. Chen, and T. W. Ling. Efficient processing of xml twig patterns with parent child edges: a look-ahead approach. In CIKM, pages 533–542, 2004.
[15] J. Lu, T. W. Ling, Z. Bao, and C. Wang. Extended xml tree pattern matching: theories and algorithms. In Technical Report, 2010.
[16] J. Lu, T. W. Ling, C. Chan, and T. Chen. From region encoding to extended dewey: On efficient processing of xml twig pattern matching. In VLDB, pages 193–204, 2005.
[17] J. Lu, T. W. Ling, T. Yu, C. Li, and W. Ni. Efficient processing of ordered XML twig pattern matching. In DEXA, pages 300–309, 2005.
[18] I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita,and C. Zhang:. Storing and querying ordered XML using a relational database system. In Proc. of SIGMOD, pages 204–215, 2002.
[19] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-friendly XML node labels. In SIGMOD, pages 903–908, 2004.

BIOGRAPHIES

Y. Ravi Kiran, he is currently pursuing M.Tech (CSE) in MLRIT, Hyderabad, AP, INDIA. He has received B.Tech in Computer Science and Systems Engineering in the year 2009 and M.B.A in finance in the year 2011. His main research interests include Data mining, Web services, XML and Big Data.

P. Dayakar, has received his M.Tech (Software Engineering) in 2010 from JNTU, Hyderabad, India, Now he is working as Assistant Professor in Dept. of CSE in MLR Institute of Technology, Hyderabad, A.P., India. He has more than 4 years' of experience in academics. He published a paper in International Journal on Network Protocols. His main research includes networks and data mining.