An Effective Algorithm for XML Tree Pattern Matching- A Holistic Approach

Mr.V.Balamurugan[1], Mr.A.Vigneshkumar[2]

[1]Assistant Professor, Department of Computer Science and Engineering, Bannari Amman Institute of Technology, Satyamangalam, Tamil Nadu, India
[2]Assistant Professor, Department of Computer Science and Engineering, Bannari Amman Institute of Technology, Satyamangalam, Tamil Nadu, India

Abstract- XML is a self-describing data representation format with a flexible structure. Since hundreds of XML-based languages have been developed, XML is widely accepted as a standard for data representation and information exchange over the internet. The major advantage of using XML is that it allows the users to create their own tags. This kind of increasing popularity of XML attracted the Business and enterprises to make queries on XML data more frequently. There is an increasing demand for efficient and effective query processing on XML data. For performing query processing operations an input XML File is required. Such an XML Files can be viewed as an XML Tree using DOM parser. XML-DOM Parser is mainly used to store, access and manipulate our XML Tree. We have proposed a new search engine named XML Search engine for pattern matching. This XML Search engine gets the input keyword query from the user and performs exact pattern matching for text, images and audio files based on an effective XML Tree Pattern Matching algorithm called TreeMatch. The downloading time of images and audio files in an XML Search engine is going to be compared with local search engine. Finally from the experimental results we are going to prove that our proposed XML Search engine performs exact pattern matching and takes less downloading time compared to local search engine.

Index Terms- XML Tree, DOM Parser, TreeMatch, TwigStack, XQuery, XPath

I. INTRODUCTION

XML is becoming a defacto standard for storing and exchanging information across various platforms. It is a hardware and platform independent tool for carrying the information. Interoperability is possible through XML. Due to the business collaborations and for the purpose of portability enterprises are storing data in XML Format. There is an increasing need for effective query processing on XML data. An input XML File is required for performing pattern matching. Such an XML Files are also validated using DTD or XML Schema. XML Parsers are available in all languages that facilitate the usage of XML programmatically. Moreover XML is tree based and it is convenient to manipulate easily using DOM (Document Object Model) API. Recently many researchers developed various methods or algorithms for processing XML tree queries. The existing system uses XML Tree Pattern Matching algorithm called TwigStack. But the major drawback of this algorithm is that it provides support only to XML Query languages like XQuery (XML Query Language) and XPath (XML Path Language). XQuery is constructed using XPath Expressions. XPath (XML Path Language) consists of lot of notations for pattern matching. But this TwigStack algorithm supports only for two notations. For performing pattern matching for P-C (Parent-Child) relationships a notation (/) is used whereas for performing pattern matching pattern matching for A-D (Ancestor-Descendant) relationships a notation (//) is used. However these kind of notations like / and //will make the query processing little bit complicated. This algorithm fails to control the size of useless intermediate results. So we have proposed an alternate solution for performing pattern matching. In our proposed system we are using keyword query and an efficient XML Tree Pattern matching algorithm called TreeMatch. This algorithm solves the sub-optimality problem faced by our existing system as it does consider useless intermediate results. This algorithm is based on the concept of Extended Dewey Labeling. According to the labeling scheme each and every node in an XML document is associated with the number or label. For an instance label 0 is assigned to the root node. The children of the root get labeling like 0.0 and continue with 0.1. The grandchildren of the first parent node starts with 0.0.0 and continue like 0.1.0. The labeling scheme makes XML Tree Pattern Matching query processing easy. Finally we are going to propose a search engine named XML Search engine which performs exact matching based on TreeMatch algorithm and takes less downloading time compared to local search engine.

II. RELATED WORK

S.Chien and C.Zhang [4] have proposed holistic algorithms for XML Query Processing. The novel holistic XML twig pattern matching method called **TwigStack** which avoids storing intermediate results unless they contribute final results. The major advantage of this method is that it avoids computation of large redundant intermediate results. But main limitation of TwigStack is that it may produce large set of "useless" intermediate results when queries contain parent child relationship. TwigStack has been proved optimal only for queries with A-D edges and it still cannot control the size of intermediate results for queries with P-D edges. TwigStack operates in two steps

1. A list of intermediate path solutions is output as intermediate results and

2. The intermediate path solutions in first step are merge-joined to produce the final solutions

Xiaoying Wu [12] have proposed **MPMGJN (Multi-Predicate Merge-Join) algorithm** and typically this algorithm consists of decomposition-matching and merging process:

- Decompose the tree pattern into linear patterns which might be binary (parent-child or ancestor –descendant) relationships between pairs of nodes or root-to-leaf paths
- Find all matching's of each linear pattern
- Merge-join them to produce results.
- MPMGJN varies from TwigStack merge join algorithm is that it requires multiple scans of input list

Li and Jaihaeng Lu [11] have proposed **Stack-Tree Algorithm** which mainly used to overcome the drawbacks of MPMGJN algorithm. The major drawback of MPMGJN algorithm is that is requires multiple scan of input list whereas Stack-Tree algorithm needs only one scan of the input lists. Stack Tree algorithm uses stacks to maintain the ancestor or parent nodes. Stack Tree Algorithm works for both P-D and A-D edges.

Jaihaeng Lu et al. [7] have proposed **OrderedTJ Algorithm** which is mainly used to overcome the drawbacks of decomposition-matching-merging algorithms. In OrderedTJ algorithm an element contributes to final results only if the order of its children accords with the order of corresponding query nodes. If we call edges between branching nodes and their children as branching edges then denote the branching edge connecting to the nth child as the nth branching edge. OrderedTJ is I/O optimal among all sequential algorithms that read entire input. In other words, the optimality of OrderedTJ allows the existence of parent-child edges in non-branching edges and the first branching edge. OrderedTJ algorithm output much less intermediate results, OrderedTJ increases linearly with the size of the database; OrderedTJ is not optimal and outputting less intermediate results.

Al-Khalifa and H.V.Jagadish [2] have proposed **TJFast algorithm** to overcome the drawbacks of containment labeling scheme. While containment labeling scheme preserves the positional information within the hierarchy of an XML Document but some limitations of containment labeling scheme are

- The information contained by a single containment label is very limited. For example, we cannot get path information from any single containment label.
- Wildcard are widely used in XPath and it cannot be supported by the containment label scheme

The containment label scheme is difficult to answer queries with wildcards in branching nodes. TJFast does not produce the individual solution for each node when there are multiple return nodes for the query. TJFast cannot work with ordered restriction and negation function.

Wen-Chiao Hsu [1] have proposed **CSI-X technique** to speed up the query evaluation in XML documents. CIS-X mainly used to overcome the drawbacks of decomposition-matching-merging algorithms to process XML Path expressions. According to decomposition-matching-merging algorithms a query is decomposed into several sub-queries, each of which is separately executed and its intermediate results stored for further processing. However these methods still have drawbacks of producing large intermediate results and time-consuming merging processing. So in this paper CIS-X technique has been proposed which support for complex XQueries. But the drawback with the CIS-X Technique is that it takes more time for index construction.

B.Choi and M.Mahoui [6] have proposed a new algorithm called **Twig Square Stack** which mainly used to eliminate the merging costs in second phase. Twig Square Stack is a one phase algorithm which can process path matching efficiently and avoids the high cost of merging phase. The overall solutions are stored in hierarchical stacks and the final solutions can be output by applying a simple enumeration function. However the data structures are too complex and expensive to maintain.

L.V.S.Lakshmanan [9] have proposed an algorithm **TwigList** which is a refined version of Twig Square Stack, utilizing a much simpler data structure, a set of lists to store solutions. TwigList has advantages over Twig Square Stack but has same shortcomings. One drawback is that all the potential nodes related to QP (Query Processing) will be pushed into and popped from the temporary stack, even though some of them are not part of the solution. Another drawback is they have less ability to efficiently discard useless nodes.

Al-Khalifa [2] has proposed **Structural Join** methods to process twig pattern matching. In the first phase, a twig query is decomposed into several binary P–C or A–D relationships. Each binary sub-query is separately evaluated and its intermediate result is produced. The final result is formed by merging these intermediate results in the second phase. This method generates a huge number of intermediate results that may not be part of the final results. In addition, the phase of merging is expensive.

T.W.Ling and C.Chen [8] have proposed a containment labeling scheme to process twig queries. Containment labeling scheme for twig Pattern processing decomposes a twig pattern into set of binary relationships which can be either P-C or A-D. Each binary relationship is processed using structural join techniques and the final match results are obtained by merging individual binary join results together. The main problem with the above solution is that it may generate large and possibly unnecessary intermediate results because the join results of individual binary relationships may not appear in the final results. This scheme still produces useless intermediate results for queries containing P-C edges and reduces the size of useless intermediate results for queries containing A-D relationships.

III. EXISTING WORKS

Existing System uses TwigStack algorithm for performing pattern matching in an XML File. TwigStack Algorithm supports XQuery and XPath only. TwigStack Algorithm provides answers to queries containing P-C and A-D relationships. P-C edges are denoted by (/) and A-D edges are denoted by (//). The TwigStack Algorithm is a decomposition-matching and merging algorithm. According to this algorithm a query is decomposed into several sub-queries. Each sub-query is executed separately and intermediate results are stored for further processing. The final result is obtained by merging these intermediate results. TwigStack Algorithm provides useless intermediate results for

queries containing P-C relationships and it controls the size of intermediate result for queries containing A-D relationship. The TwigStack algorithm is described by the following:

```
// Phase 1
1: while notEnd (q)
2: qact = getNext (q)
3: if (isNotRoot (qact)) then
4: cleanStack (parent (qact), nextL (qact))
5: end if
6: if (isRoot (qact) or isNotEmpty (Sparent (qact))) then
7: cleanStack (qact, next (qact))
8: moveStreamToStack (Tqact, Sqact, pointertotop (Sparent (qact)))
9: if (is Leaf (qact)) then
10: showSolutionsWithBlocking (Sqact, 1)
11: pop (Sqact)
12: end if
13: else
14: advance (Tqact)
15: end if
16: end while
// Phase 2
17: mergeAllPathSolutions ()
```

Algorithm **TwigStack** operates in two phases. In the first phase (lines 1-16), some (but not all) solutions to individual query root-to-leaf paths are computed. In the second phase (line 17), these solutions are merge-joined to compute the answers to the query twig pattern. The major drawbacks of an existing system are described below:

- XQuery and XPath is complicated to understand by non-database users
- XQuery and XPath are not user friendly to non-expert users
- Query Answering becomes little bit complicated using XQuery and XPath
- TwigStack Algorithm fails to control the size of useless intermediate results

IV. PROPOSED WORKS

In proposed system keyword and TreeMatch algorithm is used for performing exact pattern matching. Our input XML File is represented as a Tree using DOM Parser. An XML Search engine is created which gets the input query and performs pattern matching using an effective XML Tree Pattern Matching algorithm TreeMatch. TreeMatch algorithm is based on Extended Dewey Labeling concept. The input query matches with the Extended Dewey label and completes query processing. In proposed system we are performing pattern matching for text, images, audio and video files and the downloading time of audio and video files are computed. The downloading time of audio and video files are compared with local search engine. It is shown that XML search engine takes less downloading time. The concept of the TreeMatch Algorithm is given as follows:

```
1: locateMatchLabel (Q);
2: while(endroot)) do
3: f_act= getNext(topBranching Node);
4: if (f_act is a return node)
5: addToOutputList (NAB(f_act ,cur(Tf_act));
6: advance (Tf_act ); //read the next element in Tf_act
7: updateSet (f_act ); //update set-encoding
8: locateMatchLabel (Q); //locate next element with matching path
9: emptyAllSets (root);
```

Line 1 locates the first element whose paths match individual root-leaf path pattern. In each iteration, a leaf node f_{act} is selected by getNext function (line 3). The purpose of line 4, 5 is to insert the potential matching elements to outputlist. Line 6 advances the list Tfact and line 7 updates the set encoding. Line 8 locates the next matching element to the individual path. Finally, when all data have been processed, we need to empty all sets in Procedure EmptyAllSets (Line 9) to guarantee the completeness of output solutions.

The proposed system does not require complex query languages like XPath and XQuery. TreeMatch Algorithm matches with the extended Dewey Label for given query and then completes the query processing. Processing time of the TreeMatch Algorithm is less when compared to the decomposition-matching and merging algorithms. TreeMatch algorithm does not produce useless intermediate results. The major advantage of introducing the TreeMatch Algorithm is to solve sub-optimality problem and to reduce the answering time of the queries.

The Proposed system is implemented by using the following two main modules:

- Admin
- User

The various sub-modules used in Admin and user are given below as follows:

Admin:

1. Insertion and deletion of data
2. Creation of an XML File

User:

1. User Login
2. Viewing XML Tree
3. XML Tree Pattern Matching
4. Comparison Module

In Admin module we are creating an XML File. After the successful creation of an XML File user can login with their id and password. Users can view the XML Tree from the selected XML File. Then it is easy for the user to perform pattern matching for text, images, and audio files. Finally downloading time of audio and image files are compared with local search engine and it is proved that XML Search engine takes less time

V. EXPERIMENTS AND RESULTS

We have implemented all tested algorithms in J.D.K 1.6 using the file system as a simple storage. We conducted all the experiments on a computer with Intel Pentium IV 1.7GHz CPU and 2G of RAM. The concept of TwigStack algorithm has been tested using a Tool called XPath Builder. The Tree Match

algorithm has been implemented using Java Programming language. The front end is java and Back end is MySQL database.

All the proposed modules have been implemented and the analysis of our project is shown in the bar chart given below. In the Bar graph X-Axis is Query and Y-axis is time in milliseconds.

The pattern matching is going to be done for text, images and audio files. For performing pattern matching for images and audio files the downloading time will be the important factor. The downloading time of images and audio files in an XML Search engine is going to be compared with some local search engines. The experimental results show that our XML search engine takes less downloading time.

The major advantages of our proposed work are as follows:

- Reduced Downloading time
- Does not consider useless intermediate results
- Very low processing time
- User-friendly
- Efficient for non-data base users
-

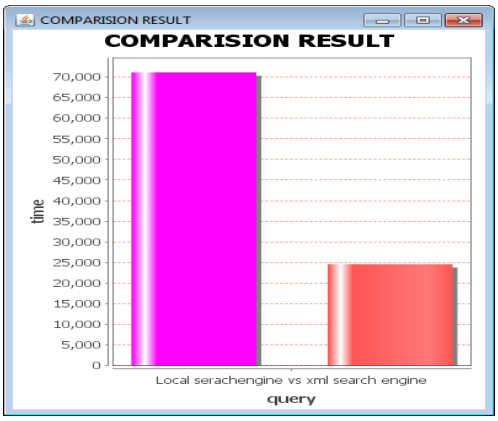

Fig 1 The downloading time of an audio file in XML Search engine is compared with Local Search engine

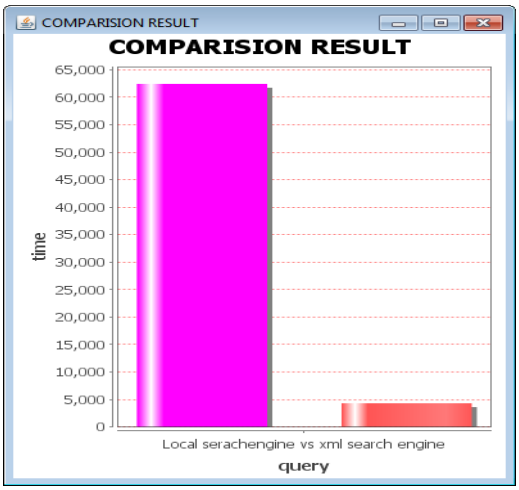

Fig 2 The downloading time of an image file in XML Search engine is compared with Local Search engine

REFERENCES

[1] Wen-Chiao, H. and Kasimoglu, S. (2013) 'An Compacted Indexing Scheme for efficient query evaluation of XML Documents' Elsevier Journal, vol. 241, no. 4, pp. 195-211.

[2] Al-Khalifa and Jagadish, H.V. (2002) 'On boosting holism in XML twig pattern matching using structural indexing techniques' Proc. of ICDE Conference, pages 141–152.

[3] Bruno, N. and Koudas, N. (2010) 'Holistic twig joins: optimal XML pattern matching' Proc. of SIGMOD Conference, pages 310–321.

[4] Chien, S. and Zhang, C. (2002) 'Efficient structural joins on indexed XML documents' Proc. of VLDB, pages 263–274.

[5] Chen Wang and Naughton, S. (2006) 'On supporting containment Queries in relational data base management Systems' Proc. of SIGMOD Conference, pages 274–285.

[6] Choi, B. and Mahoui, M. (2003) 'On the optimality of the holistic twig join algorithms' Proc. of DEXA, pages 28–37.

[7] Jiaheng Lu, Tok Wang Ling, Zhifeng Bao and Chen, W. (2011) 'Extended XML Tree Pattern Matching: Theories and Algorithms' IEEE Transactions on Knowledge and Data Engineering. Vol. no-23, no-3, pages 1-15.

[8] Ling, T.W. and Chen, C. (2007) 'On efficient processing of XML twig pattern matching' in VLDB, pages 193–204.

[9] Lakshmanan, L.V.S. (2010) 'XML Tree Pattern Processing Algorithms' Proc. Of SIGMOD conference, pages 235-241.

[10] Jaiheng Lu , Tok Wang Ling, C. (2008) 'Efficient processing of XML twig patterns with parent child edges' in CIKM, pages 533–542.

[11] Li and Jihaeng Lu, V. (2006) 'Indexing and querying XML data for regular path expressions' Proc. of VLDB, pages203-215.

[12] Muthukumaran, S. and Sudha, R. (2012) 'Efficiency of TreeMatch algorithm in XML Tree Pattern Matching' International J. of Computer Science and Software Technology, Volume 4, Issue 5, pages 19-36.

[13] Ravi Kumar, D. and Ramadevi, S. (2013) 'Data Mining for XML Query Answering Support' IOSR Journal of Computer Science, Volume 2 Issue 5, pages 25-30.

[14] Saravana Kumar, D. and Madhu, S. (2012) 'Efficient Handling of XML Tree Pattern Matching Queries-A holistic Approach' International Journal of Advanced Research in Computer and Communication, Volume 1 Issue 5, pages 530-534.

[15] Sivarama Prasad, M. (2011) 'A survey of Algorithms Related to XML Based Pattern Matching' International J. Engineering and Technology, Vol 2. Issue 9 pages 1028-1032.

[16] Satya Sandeep, S. (2013) 'A new Holistic algorithm and Application for XML Tree Pattern Matching' International J. of Computer Technology, Volume 3 Issue 5, pages 1782-1788.

[17] Suresh Babu, D. and Shiva, B. (2012) 'Extended XML Tree Pattern Matching using TreeMatch Algorithm' International J. of Computer Science and Information Technology, Vol.3 Issue 5. Pages 5210-5215.

[18] Xiaoying, W. (2008) 'An efficient XML Tree Pattern Matching Algorithms' Proc. of VLDB, pages 114-130.

[19] J.D. Ullman, Principles of Database and Knowledge-Base Systems, vol. 1. Computer Science Press, 1988.

[20] S. Abiteboul and V. Vianu, "Queries and Computation on the Web" Theoretical Computer Science, vol. 239, no. 2, pp. 231-255, 2000

AUTHORS

First Author – The author Mr.V.Balamurugan1 is currently working as an Assistant Professor in the Department of Computer Science and Engineering in Bannari Amman Institute of Technology. He has completed his UG & PG in Anna University, Chennai. He has published 6 Research papers in Highly Impact factor Journals. He has published 2 papers in International Conference and 4

papers in National conference. He is a member of various professional bodies like ISTE, CSI, IAENG, UACEE and SAI. His area of research interests are Web Technology, XML and Web Services, Data Mining and Data Structures. He is one of the university rank holder during the Academic year 2012-2014. He has attended 4 workshops and seminars.

Second Author – The author Mr.A.Vigneshkumar2 is currently working as an Assistant Professor in the Department of Computer Science and Engineering in Bannari Amman Institute of Technology. He has completed his UG & PG in Anna University, Chennai. He has published 1 paper in International Conference and 2 papers in National conference. His area of research interests are Computer Networks, Cloud computing, big data, XML and Web Services and Data Mining. He has attended 3 workshops and seminars.