ISSN:0975-9646

A Secured Wireless XML Streaming Supporting Twig Pattern Queries

K. Preethi[1], S. Ganesh Kumar[2]

[1]MTech Student, [2]Assistant Professor
Department of Computer Science and Engineering
SRM University, Kattankulathur, Tamil Nadu, India

***Abstract*— The migration to wireless network from wired network has been a universal trend in recent days. Wireless broadcasting is a mechanism for dispersing identical information in the wireless mobile environment. In this paper, we propose an XML broadcasting scheme. The wireless XML data stream consists of sequence of integrated nodes, called G-node. G-node is a unit structure which consists of node name, location path, child index, attribute index, lineage code, Attribute Value List and Text List. An encoding scheme, called Lineage Encoding, is defined to represent the parent-child relationships among XML elements as a sequence of bit-strings, called Lineage Code (V, H). The components of G-node are used to process twig pattern queries at mobile clients efficiently. Twig pattern query involves element selections satisfying complex patterns in tree-structured XML data. Symmetric encryption technique is used for ensuring security at mobile clients. By adopting these techniques mobile clients can retrieve the required data satisfying the given twig pattern query and security is provided so that one mobile client cannot view other mobile clients information.**

***Keywords*— Wireless Broadcast, XML Data Stream, Lineage Encoding, Twig Pattern Query, Symmetric Encryption Technique.**

I. INTRODUCTION

Increased use of laptop computers within the enterprise, and increase in worker mobility has increased the demand for wireless networks. The mobility, scalability and ease of physical setup brought by wireless network have made it possible in many applications when compared to wired networks. With the rapid growth of this technology wireless mobile computing has become more prominent. User mobility and device portability are two main aspects of mobile computing. With the rapidly expanding technology of laptops and smart phones, users can communicate and access information from anywhere and at any time. Wireless broadcasting is an effective information broadcasting method from a server to a pool of clients in the wireless environment. We propose an energy efficient wireless XML streaming scheme supporting twig pattern queries in the wireless mobile environment. XML is used to structure the data and also provide meaning for data. The popularity of XML is increasing rapidly and more and more information is being stored, exchanged and presented in XML format. A sample XML document is given in Fig. 1.

We define a novel unit structure called G-node for streaming XML data in the wireless environment. We define an encoding scheme, called Lineage Encoding, to represent parent-child relationships among XML elements and to support twig pattern queries. The components of G-node enable mobile clients to download relevant data during query processing. Efficient query processing over XML data is more important. In this paper, we use XPath as the query language. An XML twig pattern query, represented as a small query tree, is essentially a complex selection on the structure of an XML document. Symmetric encryption technique is used for ensuring security at mobile clients. In the client-side, if a query is issued by the mobile client, the mobile client tunes in to the broadcast channel and selectively downloads the relevant data.

```
<Studentinfo>
<student id =”1015511”>
<personalinfo id=”2015511”>
  <studname id=”3015511”personalinfo=”2015511”>
    <firstname id=”4015511”> preethi </firstname>
    <lastname id=”4015512”> k</lastname>
  </studname>
  <rollno id="3015512" personalinfo=”2015511”> 11</rollno>
  <email id=”3015513” personalinfo=”2015511”> preethi@gmail.com
  </email>
</personalinfo>
<result id=”2015512”>
  <subject1 id=”3015514” name=”maths” student=”1015511”
  result=”2015512”>
    <theorymark id=”4015513”>80</theorymark>
    <pracmark id=”4015514”>20</pracmark>
  </subject>
  <subject2 id=”3015515” name=”science” student=”1015511”
  result=”2015512”>
     <theorymark id=”4015515”>70</theorymark>
    <pracmark id=”4015516”>20</pracmark>
  </subject>
</result>
</Student>
</studentsinfo>
```

Fig. 1 A Sample XML Document

II. BACKGROUND

A. XML Data Model

XML documents have a hierarchical structure and can be represented as a rooted, ordered, and labeled tree. These XML trees (twigs) are available in two forms; they are ordered and unordered XML trees. The present approach considers an ordered and labeled XML tree. XML document must contain a root element. This element is the parent of all other elements. All elements in an XML document can contain sub elements, text and attributes. The tree represented by an XML document starts at the root element and branches to the lowest level of elements. The nodes of the XML tree represent elements and the edges represent parent-child relationships among XML elements. The XML tree representation for the sample XML document is given in Fig. 2. In this paper, we use XPath [1] as the query language. The XML query language namely XPath represent the query as ordered labeled small trees (twigs). A twig pattern query consists of two or more path expressions and it represents complex search condition. For example, twig pattern query written in XPath format and the tree representation is given in Fig. 3.

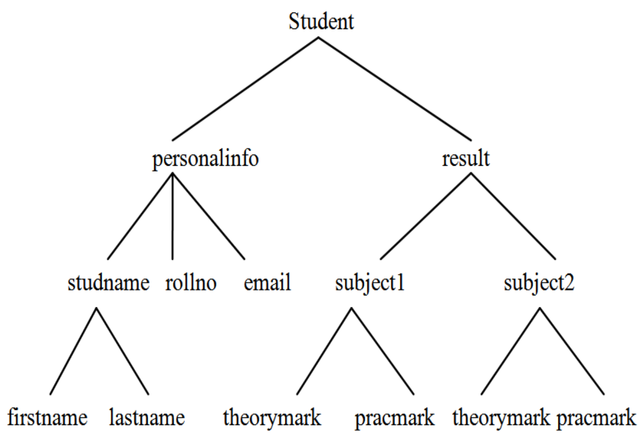

Fig. 2 XML Tree Representation

Q1: //studentinfo/student [name/text () =”predicate”]/result

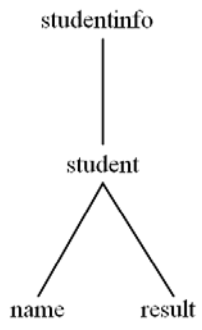

Fig. 3 Example twig pattern query and its tree representation

B. Twig Pattern Query

A twig pattern query involves element selections satisfying complex patterns in a tree-structured XML data. It consists of two or more path expressions and it supports all sorts of complex queries [2], [3], [4], [5]. Many technique uses structure index which directly captures the structural information of XML document is used for XML query processing [8], [9]. This approach reduces the size of XML data stream by integrating indexes of multiple elements of the same path into one node. But they do not preserve all parent-child relationship and cannot find the exact output elements. The twig pattern query which uses tree structures finds the exact output elements and supports complex query patterns. The twig pattern query decomposes the tree pattern into linear pattern which can be binary (parent-child or ancestor-descendant) relationships between pairs of nodes. Then it finds all matches of each linear pattern and joins them to produce the result. Thus the twig pattern query processes.

III. Related Works

A. Multipredicate Merge Join Algorithm

Multipredicate merge join (MPMGJN) algorithm [5] uses a merge join algorithm to provide higher cache utilization and superior performance than a standard RDBMS algorithm. The inverted list engine uses a merge join (MPMGJN) as its join operator. This algorithm is different from the standard merge join and the index nested-loop join algorithms, and the difference has a significant impact on performance. The MPMGJN algorithm out-performs the standard RDBMS join algorithms by more than an order of magnitude on containment queries. Containment queries are a class of queries based on containment relationships among elements, attributes, and their contents. The merge join algorithm used in the inverted list engine uses all join columns to guide merging and thus it avoids some row comparisons done by the standard merge join. But the standard merge join is only one of the choices of an RDBMS. Two other algorithms namely hash join and index nested-loop join can be used to process a join. Since a hash join cannot be used for inequality predicates, only the predicate on a particular field can be used, and the inequality predicates must be applied on each pair of rows just like the standard merge join. Therefore, hash join has the same disadvantage. In index nested- loop join, for each outer row, its values are used to seek the index on the inner table, starting from the root of the tree and reaching a record with the start key at the bottom of the index. An index scan is then conducted across the index records until one with a stop key is reached. Then each record along the scan is attempted to be joined with the outer row. The seeking and scanning are repeated for all outer rows. Sometimes it appears that the standard index nested-loop join does fewer comparisons than MPMGJN, and therefore should perform better. But this is not always true. In order to selectively examine inner rows, an index must be used, and comparisons must be done on index records. But experimental result shows that an index record comparison almost always incurs a cache miss. A merge join (MPMGJN) is essentially a form of nested-loop join, except that seeking is not done on an index, but rather directly on data records. For the same query on the same data, record scans cost the same number of comparisons as index scans, but the record seek costs are different from the index seek costs. Therefore MPMGN outperforms well and has better cache utilization.

B. Structural Joins

Al-Khalifa et al. [2] proposed two families of structural join algorithms for efficient XML query processing. For this task, tree-merge and stack-tree algorithm was proposed to find all occurrences of parent-child and ancestor-descendant structural relationships in an XML database. The tree merge algorithm is a natural extension of merge join to deal with the multiple inequality condition that characterizes the parent-child or ancestor-descendant structural relationships based on the representation. But the stack tree algorithm has no counterpart in traditional relational join processing. Structural join algorithms are more efficient than traditional join algorithms implemented in relational databases for the same task The stack-tree algorithm is I/O and CPU optimal and has worst case linear I/O and CPU complexities for both parent-child and ancestor-descendant structural relationships. The tree-merge algorithm has worst case quadratic I/O and CPU complexities but in some cases they have linear complexities. Pointer based joins which are useful for query processing is not addressed in this paper and many issues are yet to be explored.

C. Holistic Twig Joins

Nicolas et al. [6] proposed twig join algorithms namely path stack and twig stack for matching XML twig pattern query. PathStack algorithm decomposes the twig into multiple root-to-leaf path patterns and identifies solutions to each individual path. It then merge these solutions to compute answers to the query. But using the PathStack is suboptimal since many intermediate solutions do not contribute to the final answer. The sub optimality problem is overcome by the TwigStack. TwigStack reduces the amount of the intermediate results and computational cost for merging the intermediate results using a chain of linked stacks that represent partial results to root-to-leaf query path. When the twig pattern uses only ancestor-descendant relationships between elements, TwigStack is I/O and CPU optimal among all sequential algorithms. Therefore TwigStack is I/O and CPU optimal for a large class of twig pattern queries and practically efficient. But there is more to efficient XML query processing than is within the scope of this paper.

D. XML Twig Queries with OR-Predicates

Previously proposed twig pattern matching algorithms deal with only twig queries without OR-predicates. A novel holistic-processing algorithm for twig queries with OR-predicates is used. The twig pattern query is processed without decomposing it. In general the existing algorithms follows a straightforward approach that decomposes a twig query with OR-predicates into multiple twig queries without OR-predicates and then it combines their results is obviously not optimal in most cases. In [7], Jiang et al. use two algorithms, a merge-based algorithm for sorted XML data and an index-based algorithm for indexed XML data, to enhance performance for matching twig queries with the OR-predicate. Holistic processing is much more efficient than the decomposition approach. Furthermore, the use of indexes skips elements during a join and significantly improves the performance for matching twig queries with OR/AND-predicates. But indexing method is so far not efficient for twig pattern queries.

E. Integration of Structure Indexes and Inverted Lists

Kaushik et al. [9] proposed an XML path query processing algorithm integrating inverted lists and structure indices. Structure indexes are used as a substitute for graph traversal. These structure indexes are proven to be very effective when applied to queries that examine the "coarse" structure of documents. But the structure indexing approach is much less successful when we consider queries on "values" or text words in the documents. On the other hand, while inverted lists have proven very effective for keyword searches in the information retrieval community, when applied to path expression queries over XML documents they are less effective. The problem is that evaluating a path may require many joins over large inverted lists, and these joins can be expensive. This paper proposes a strategy that combines structure indexes and inverted lists, and a query evaluation algorithm for branching path expressions based on this strategy. This technique reduces computation costs omitting join operations. But this technique does not support complex query languages and more complex ranking functions. Beyond that, the problem of running structured queries over hyper-linked XML documents needs to be addressed.

IV. Wireless XML Streaming

A. XML Stream Generation

For providing energy efficient query processing in the wireless and mobile environments, wireless XML stream generation approach has been proposed. In this approach, the XML data are streamed in the wireless environment by the server. The XML document comprises of student (client) details. Wireless XML stream is generated by using SAX parser. SAX parser is a simple API for XML. This approach has been proposed to reduce structural overheads of the original XML document and indices containing timing information are attached to the XML data stream. These works enable mobile clients to download and view their data from XML data stream. We define a novel unit structure called G-node for streaming XML data in the wireless environment. The G-node is the basic element of the wireless XML stream. The G-node comprises of node name, location path, lineage codes, child index, attribute index, AVL, and TL. Node name is the tag name of the elements, and location path is the path of the element from the root node. Child Index contains child names and addresses that point to the starting position of child G-nodes in the wireless XML stream. Attribute Index contains attribute names and addresses that point to the starting position of the values of the attribute that are stored in Attribute Value List. Text Index is an address pointing to the starting position of Text List. Attribute Value List (AVL) and Text List (TL) store attribute values and text contents of the element. In our scheme, we exploit the benefits of the Attribute Summarization technique [10] to reduce the size of a wireless XML stream. In XML, an element may have multiple attributes, each of which consists of a name and value pair. In addition, there is a structural characteristic that elements with the same tag name and location path often contain the attributes of the same name. Attribute Summarization technique eliminates repetitive attribute names in a set of elements when generating a stream of G-nodes. The components of G-node are used to efficiently process queries in the mobile clients.

Symmetric encryption technique (AES) is used to encrypt the attribute values and text contents of each element excluding name and roll number element in the XML document. Advanced Encryption Standard (AES) algorithm is used to ensure security in the mobile clients. The risks to users of wireless technology have increased as the service has become more popular. Despite the convenience and advantage that wireless network offers, the network can be hacked. Denial of Service, Spoofing, and Eavesdropping are some of the important threats. To overcome from these threats, and to ensure privacy and security in the mobile clients, encryption technique has been proposed. The attribute values and text contents of each student element are encrypted using symmetric encryption technique by the server and stored in AVL and TL. The decryption key of each (student) client is intimated to the mobile clients. The Symmetric encryption technique prevents unauthorized clients from viewing the data.

B. Lineage Encoding

In this paper, we propose an encoding scheme, called Lineage Encoding, to support twig pattern queries. The Lineage Encoding scheme represents the parent-child relationships among XML elements in the XML document. We propose two kinds of lineage codes, i.e., vertical code denoted by Lineage Code (V) and horizontal code denoted by Lineage Code (H). Lineage code (V) and Lineage code (H) represents the parent-child relationships among XML elements as a sequence of bit-strings. Assume that a G-node N is a child of G-node M. If the elements in En of G-node N maps to the elements in Em of G-node M, then the Lineage Code of G-node N is defined by LC (V, H), where LC (V) is a bit string and LC (H) is an ordered list of positive integers. Lineage code (V) will be set to 1 if the elements in Em has at least one child element in En else it will be set to 0. Each positive integer in Lineage code (H) denotes the number of child elements in En mapped to the same parent element in Em. Thus, the proposed Lineage Code scheme is a light weight scheme which represents parent-child relationships between elements in the XML document.

V. Twig Pattern Query Processing

The ability to efficiently retrieve required data from XML data sources is more important. Matching twig queries is a core operation in XML query processing. Twig pattern query processing consists of three phases namely tree traversal phase, sub path traversal phase, and main path traversal phase. While processing a twig pattern query with predicates, we should select subset of elements satisfying the given predicates. Then, for the selected elements, we should find their parent elements or child elements. For example, to process the query in Fig. 3, Q1://studentinfo/student[name/text()=”predicate”]/result, we should find a subset of “name” elements satisfying the given predicate condition, select their parent “student” elements, and then identify “result” elements which are children of those “student” elements. A subset of the elements selected in a G-node can be represented by a bit string, called a selection bit string (SB) for the G-node, where 1-value bits identify the selected elements. First, a function to obtain a selection bit string identifying a subset of elements in a particular child G-node is defined as follows:

Function 1:

$$Vp = Shrink\&Mask(V, SBm)$$

$$SBn = Unpack(Vp, H)$$

A selection bit string SBn for child G-node N can be computed based on the Lineage Code of N, (V, H), using Shrink&Mask and Unpack operators in order. $\text{Shrink\&Mask}(\text{V}, \text{SBm})$, where V denotes LC (V) of child G-node and elements in M with one or more child elements in N are selected by SBm. Shrink&Mask operator computes Vp by shrinking 0’s in LC (V) and then it shrinks SBm by eliminating the bits in same position as those removed in V. $\text{Unpack}(\text{Vp}, \text{H})$, where Vp is the shrunken bit string computed by Shrink&Mask operator and H denotes LC (H) of child G-node. Unpack operator extends Vp based on H to obtain the result selection bit string for the G-node N. Thus a subset of elements in a particular child G-node is selected. Second, a function to identify the parent elements of a subset of elements selected in a G-node is defined as follows:

Function 2:

$$Vp = Pack(SBn, H)$$

$$SBm = Expand\&Mask(V, Vp)$$

To identify the parent elements of a subset of elements selected in a G-node, Pack and Expand&Mask operators are used. Pack(SBn, H) operator computes Vp by shrinking the bit string SBn based on H. Vp denotes the elements in the parent G-node of N which are parents of the elements in N selected by SBn. Expand&Mask (V, Vp) operator expands Vp and masks V with it to obtain the result selection bit string SBm for parent G-node of N. Thus selection bit string to identify a subset of elements in a particular child G-node and to identify the parent elements of a subset of elements selected in a G-node is found. Finally, we define a function GetSelectionBitString (J) to select elements in a G-node contained in the query tree of a given twig pattern query, which satisfy all the branching paths and predicate conditions in the sub-tree. The selection bit string SBj for J can be computed by performing bitwise AND operations over all the selection bit strings SBm obtained from the child nodes of J where J is a G-node in the query tree T.

A. Sub path and Main path Traversal Phase

The main path denotes a path from the root node to a leaf node which represents the target element of the query and the sub path denotes branch paths excluding the main path in the query tree. The mobile client enters query and decryption key into the application. The query is then modeled into a query tree. In the tree traversal phase, the query tree is traversed in a depth-first manner; it selectively downloads components of the relevant G-nodes into the nodes in the query tree. Attribute values and texts involved in the given predicates are decrypted using the decryption key and downloaded into the relevant nodes. In the Subpaths traversal phase, the mobile client performs a post-order depth-first traversal starting from the highest branching node in the query tree using the GetSelectionBitString () function. In the sub path traversal phase each sub path is explored from the leaf node. Thus, the selection bit string for the branching node is calculated from all the sub paths in a bottom-up manner using Pack and Expand&Mask operators. Finally, the Main path traversal phase propagates the selection bit string on the branching node along the main path using Shrink&Mask and Unpack operators. Finally, the mobile client retrieves the required data which satisfies the given twig pattern query and with the help of encryption technique

VI. Conclusion

Twig pattern queries containing complex conditions are popular and critical in XML query processing. In this paper, we propose an efficient wireless XML streaming method supporting twig pattern queries. We defined Lineage Encoding scheme and relevant operators to efficiently process twig pattern queries and for selective access of XML elements as well as their attribute values and text. We also propose symmetric encryption technique to encrypt the attribute values and text contents of each element excluding name and roll number element in the XML document which ensures privacy and security in the mobile clients. Thus the mobile client can retrieve the required data satisfying the given twig pattern query and with the use of AES algorithm, one mobile client cannot view other mobile clients information.

Acknowledgment

I would like to express my gratitude and sincere thanks to my guide Professor Mr. S. Ganesh Kumar who is providing constant encouragement and generous assistance. I am heartily thankful to him for his constant support and guidelines.

References

[1] A. Berglund, S. Boag, D. Chamberlin, M.F. Fernandez, M. Kay, J. Robie, and J. Simeon, “XML Path Language (XPath) 2.0,” Technical Report W3C, 2002.

[2] S. Al-Khalifa, H.V. Jagadish, N. Koudas, J.M. Patel, D. Srivastava, and Y. Wu, “Structural Joins: A Primitive for Efficient XML Query Pattern Matching,” Proc. Int’l Conf. Data Eng. (ICDE), pp. 141-152, Feb. 2002.

[3] S. Amer-Yahia, S. Cho, L.V.S. Lakshmanan, and D. Srivastava, “Minimization of Tree Pattern Queries,” Proc. ACM SIGMOD Int’l Conf. Management of Data Conf., pp. 497-508, 2001.

[4] I. Tatarinov, S. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, “Storing and Querying Ordered XML Using a Relational Database System,” Proc. ACM SIGMOD Conf., pp. 204-215, 2002.

[5] C. Zhang, J.F. Naughton, D.J. DeWitt, Q. Luo, and G.M. Lohman, “On Supporting Containment Queries in Relational Database Management Systems,” Proc. ACM SIGMOD Int’l Conf. Management of Data Conf., pp. 425-436, 2001.

[6] N. Bruno, D. Srivastava, and N. Koudas, “Holistic Twig Joins: Optimal XML Pattern Matching,” Proc. ACM SIGMOD Int’l Conf. Management of Data Conf., pp. 310-321, 2002.

[7] H. Jiang, H. Lu, and W. Wang, “Efficient Processing of XML Twig Queries with OR-Predicates,” Proc. ACM SIGMOD Int’l Management of Data Conf., pp. 59-70, June 2004

[8] R. Kaushik, P. Bohannon, J.F. Naughton, and H.F. Korth, “Covering Indexes for Branching Path Queries,” Proc. ACM SIGMOD Int’l Management of Data Conf., pp. 133-144, June 2002.

[9] R. Kaushik, R. Krishnamurthy, J.F. Naughton, and R. Ramakrishnan, “On the Integration of Structure Indexes and Inverted Lists,” Proc. ACM SIGMOD Int’l Management of Data Conf., June 2004.

[10] J.P. Park, C.-S. Park, and Y.D. Chung, “Attribute Summarization: A Technique for Wireless XML Streaming,” Proc. Interaction Sciences, pp.492-496,Dec.2009.