DEPARTMENT – STUDENT LIBRARY USING TWIG PATTERN QUERY PROCESSING OVER ADMIN-USER LOGIN PRIVILEGE

1ALBERT MAYAN J., 2SURYA, B., 3PRANOY PRABHAKAR, 4PRINCE KUMAR

Department of Computer Science and Engineering, Sathyabama University, Chennai, India
albertmayan@gmail.com, 2uryasrinii13@gmail.com, 3pranoyprabhakar2009@gmail.com,
4prineraji281292@gmail.com

ABSTRACT
An xml twig pattern query processing library, an energy and latency efficient XML dissemination scheme for the system. Wireless broadcasting is an effective information dissemination approach in the wireless environment because of the following benefits. The server can support a massive number of clients without additional costs. The broadcast channel is shared by many clients and the clients can receive data without sending request messages that consumes energy. In Wireless XML broadcasting, the broadcast server retrieves XML data to be disseminated from the repository. An XML stream is organized to enable a selective access scheme for simple XPath queries, by borrowing the pattern summary technique, which was originally devised for indexing semi-structured data. Using Lineage Encoding, to support evaluation of predicates and twig pattern queries over the stream.

Keywords: Twig pattern, Shredded Data , XFilter , Tuples, pureXML.

1. INTRODUCTION
In the recent years, business and enterprises generate and exchange XML data more often. The XML data can be very complex and deeply nested. Therefore, there is a lot of interest in query processing over data that conforms to a tree-structured data model. Efficiently matching all twig patterns in an XML database is a major concern of XML query processing. The elements in the document can be inserted, deleted and modified. Consequently, multiple versions of the document exist. Document authors and version management systems stand to benefit from being able to manage this historical information efficiently.

This includes the ability to store, retrieve and perform queries that reveal how the document has changed over time. In order to avoid the loss of XML content, an obvious solution is to store each version of the document together with a timestamp successively in a repository or as a collection of flat files. However, both these approaches are inefficient in utilizing storage capacity and would have difficulty allowing queries on the history and evolution of a document. A more tangible solution uses a database for the storage of a single full copy of the document and subsequent changes that were made. The important contribution to XML tree pattern query minimization, proposed unique approach for tree pattern query minimization that exploits broad constraints/semantics available with XML data and renders great opportunities for thorough minimization of XML queries before further processing.

The implementation algorithms he designed is theoretically optimal (in time and space), and experimentally confirmed. Practical XML queries often require full support for more general twig patterns processing phases, Tree traversal phase, Sub paths traversal phase, Main path traversal phase.

2. BACKGROUND AND RELATED WORKS
Conventional XML query processing methods are inefficient in the Wireless mobile environment due to their huge indices. The typically use simple keyword matching or “bag of words” Information. Retrieval (IR) techniques to represent user profiles and match them against new data items. These techniques indexing mecha-nism didn’t consider network failures such as tail drops and packet losses.

Communication is not stable in Wireless broadcasting environment. Current SDI systems, however, often suffer from limited ability to express user interests, thereby raising the potential that the users receive irrelevant data while not receiving the information they need. XML query languages are complex, with many features. Natural and pervasive operation: matching XML data with a tree structured pattern[1]. Efficiently matching all twig patterns[2,4,6] in an XML database is a major concern of XML query processing. Specific twig patterns[5] can match relevant portions of the XML database. Decompose query into binary relationships. Solve binary joins against XML database. Combine together “basic” matches, its otherwise known as twig joins[3].

Normally an XML document is modeled as an ordered tree, without considering ID references. (ignoring node labels)- the tree structure of a bookstore document. In an XML tree, the internal nodes represent the elements and attributes in the document, and the leaf nodes represent the data values which are either a text in an element or an attribute value. Thus a node name is a tag label, an attribute name or a value. Edges in an XML tree reflect element-sub element, element-attribute, element-value, and attribute-value pairs. Two nodes connected by a tree edge are in parent-child (PC) relationship, and the two nodes on the same path are in ancestor-descendant (AD) relationship. The core query pattern in most standard XML query languages (e.g., XPath and XQuery) is also in a tree-like structure, which is often referred as a twig pattern.

In particular, an XPath query is normally modeled as a twig pattern query, and an XQuery query is normally modeled as several twig patterns linked by joins. For example, the XPath query Q1 i.e.,/*subject/book/title[="Network"]/name , can be represented as a twig pattern querying Fig. 1.

Fig. 1: Twig Pattern Querying

Motivation
The structural join based approaches are proven efficient in structural search. However, because they do not consider the semantics of value and other types of document nodes, they suffer from several problems during query processing.
Inverted list management

In most structural join based approaches, all nodes including elements, attributes and values in an XML tree are labeled and the labels of each type of nodes are organized in an inverted lists. When we build inverted lists for values, the number of different values causes a problem of managing a tremendous number of inverted lists. Based on our investigation, a 100MB XML document contains over 300 thousand different values, which correspond to 300 thousand inverted lists. This number will linearly increase according to the document size increase.

Advanced content search

Since twig pattern query normally models XPath expression, the advanced content search, such as numeric range search, containment search or even conjunction/disjunction of several value comparisons, which often appear in XPath query predicates may also appear as a leaf node in a twig pattern query. Without handling values specially, existing approaches have difficulty in supporting these advanced content search.

For example, to process a query to find the books with the price greater than 15, it is time consuming to get all the inverted lists with the numeric names greater than 15, and combine labels in them by document order, to perform this range search. Also structural join with inverted lists can hardly support containment search, such as /book[contains(@title, "XML query")]/price.

Redundant search in inverted lists

Inverted lists for values do not have semantic meanings. This may cause redundant search during inverted list scanning. For example, when a query is interested in books with the price of 35 in the bookstore document, structural search scans the inverted list for the value node "35" (denoted by T 35 ). Since in T 35 we do not differentiate whether a label corresponds to price or quantity, we need to check all labels in this inverted list though many of them stand for quantity of 35, and definitely do not contribute to the query result.

Actual value extraction

To answer a query, what we need is not twig pattern occurrences represented as tuples of labels, but value results. For example, after finding a number of occurrences of the twig pattern query in Fig. 2(a), we need to know the value under each name node. One major advantage of the structural join based approaches is that they only need to load relevant inverted lists to process a query, instead of scanning the whole document with high I/O cost.

However, when a query asks for values of a certain property, after getting a set of resulting labels of that property from pattern matching, they cannot find the child value under each label using inverted lists. To extract actual values, they have to read the document again, which violate the initial objective in I/O saving. Motivated on solving all these problems, we propose a semantic approach that uses both inverted lists and relational tables to perform twig pattern matching.

2. PROPOSED-SYSTEM

XML is rapidly becoming a popular data format. It can be expected that soon large volumes of XML data will exist. Use containment relationships between query nodes. XML data is either produced manually or it is generated by a new generation of software tools for the WWW and/or electronic data interchange (EDI). Exploit indexes. Skip irrelevant document fragments. We have developed a document filtering system, named XFilter that provides highly efficient matching of XML documents to large numbers of user profiles. In XFilter, user interests are represented as queries using the XPath language.

The XFilter engine uses a sophisticated index structure and a modified Finite State Machine (FSM) approach to quickly locate and examine relevant profiles. We have to consider many issues like depth-first traversal of elements increases the access time for specific queries and issues in indexing mechanism. We have to consider network failures such as tail drops and packet losses. We evaluated the performance of our scheme in the experiments, compared not only to the previous wireless XML streaming methods but also conventional XML query processing methods supporting twig pattern matching.

4. SYSTEM-ARCHITECTURE

The system architecture (fig.2) – processed as:
- An user/browser/client application accessing the xml database by sql (sql language)/is query processed over XML SQL Utility (XSU) towards xml file data base where pattern matching over query test and I/OBs take place within the xml database
- From xsu towards browser application as dom or string
- From xml file to browser application as XML doc from lob
- From query processing to browser application passing and connected through twig phase pattern –xml paser and XSD
- The Xml, which is connected towards browser application ,is connected with xsl-t processor over xsl style sheet
- Twig join algorithm was connected directly towards browser application
- The process from the xml to browser application was under twig merge
- Hence from the client request is processed and required response is obtained from the server or browser application.

Keywords Explanation:
XML SQL Utility (XSU) is an XDK component that enables you to transfer XML data through Oracle SQL statements. You can use XSU to perform the following tasks: Transform data in object-relational database tables or views into XML. XSU can query the database and return the result set as an XML document.
- Extract data from an XML document and use canonical mapping to insert the data into a table or a view or update or delete values of the appropriate columns or attributes.

XML Parsing- Parsing XML refers to going through XML document to access data or to modify data in one or other way.
XML Parser provides way how to access or modify data present in an XML document. Java provides multiple options to parse XML document. Following are various types of parsers which are commonly used to parse XML documents:

- **Dom Parser** - Parses the document by loading the complete contents of the document and creating its complete hierarchical tree in memory.
- **SAX Parser** - Parses the document on event based triggers. Does not load the complete document into the memory.
- **JDOM Parser** - Parses the document in similar fashion to DOM parser but in more easier way.
- **StAX Parser** - Parses the document in similar fashion to SAX parser but in more efficient way.
- **XPath Parser** - Parses the XML based on expression and is used extensively in conjunction with XSLT.
- **DOM4J Parser** - A java library to parse XML, XPath and XSLT using Java Collections Framework, provides support for DOM, SAX and JAXP.

**Query-processing**

- The retrieval of information from a database according to a set of retrieval criteria, the database itself remaining unchanged.
- In the context of a specific query language, the technique of translating the retrieval criteria specified using the language into defs. (1) into more primitive database-access software, including a selection among different methods to choose the most efficient in the particular circumstances.

LOBs are designed to support the unstructured data.

**DOM-String** is a UTF-16 String. As JavaScript already uses such strings, DOM-String is mapped directly to a String. An XML Schema describes the structure of an XML document. The XML Schema language is also referred to as XML Schema Definition (XSD).

**TWIG JOIN** is the merged or joined search results of new and the related old search results. (by using twig pattern and centralized pattern search-twig stack).

**XSLT** (Extensible Style sheet Language Transformations) is a language for transforming XML documents into other [1] publishing must join all 12 relational tables and construct the XML document. Queries Q1 (Select*) and Q3 XML documents, or other formats such as HTML for (5PredSome) return many documents, so the cost for web pages, plain text or into XSL Formatting Objects, which may subsequently be converted to other formats, such as PDF, PostScript and PNG. In computing, the term Extensible Style sheet Language (XSL) is used to refer to a family of languages used to transform and render XML documents. Historically, the XSL Working Group in W3C produced a draft specification under the name XSL, which eventually split into three parts:

- **XSL Transformation (XSLT):** an XML language for transforming XML documents
- **XSL Formatting Objects (XSL-FO):** an XML language for specifying the visual formatting of an XML document
- **XML Path Language (XPath):** a non-XML language used by XSLT, and also available for use in non-XSLT contexts, for addressing the parts of an XML document.

5. **COMPARING XML QUERIES ON SHREDDED DATA TO PURE XML**

Assume you need to serve an XML-oriented application or web service that requires query results in XML format. If the XML data is shredded to relational tables, relational query results must be converted back to XML. For this purpose, SQL/XML publishing functions can be used in the SELECT clause of SQL queries to construct the required XML tagging for the result set. Five SQL queries were defined over the relational schema, that are logically equivalent to our five XQueries over XML columns.

These SQL queries use traditional relational predicates, join some or all of the tables, and use SQL/XML publishing functions to return the same XML results as the XQueries. Query Q2 (1Pred1Doc) returns only one document, based on an account number lookup. In both cases (pure XML and publishing), there are indexes on the account number to improve the performance of this query. Although the SQL/XML query is quite fast, it's still 75 times slower than an XQuery on an XML column. This is because an XQuery only needs to locate the document through the index and serialize it, while SQL/XML SQL/XML document construction from relational data is multiplied. Query Q4 (PartialAll) reads a few values from each XML document and constructs a whole new document out of them.

With XQuery over pureXML storage the values are read from the XML column and the construction is expressed in XQuery. With SQL/XML publishing over shredded data, the values are read from relational columns, and the construction is expressed in SQL/XML. In both cases, the construction is the bottleneck. This is why both versions of Q4 perform approximately identically. Q5 (PartialSome) uses multiple predicates to return very few XML documents with only one element. In this case, SQL/XML is not very expensive because it constructs just one element per result row and joins only 3 of the 12 tables. Since searching over relational data can be somewhat faster than searching over XML data, you see better performance for this query over shredded data (20 times faster than XQuery in our setup).

6. **TWIG QUERY PROCESSING AND SEARCHING**

Twig pattern contains three steps:

a) content search and query rewriting.

b) structural search.

c) value extraction.

Theoretically, the first two steps, i.e., content search and structural search, can be reordered. The reason why we perform content search before structural search is that content search normally results in high selectivity. By performing content search first, we can reduce the complexity of structural joins. This is similar to selection push ahead in relational query optimizers.

The pseudo-code of **Twig query processing** is presented as in the Algorithm.

1: // step 1: perform content search, construct new inverted lists and rewrite the query
2: while there are more value comparisons in predicates of Q do
3: let c be the next value comparison, and p be its property
   (parent element or attribute)
4: create a new inverted list Tp for p
5: select the labels based on c from the table Rp, and sort
   the resulting labels by document order
6: put the selected labels into Tp
7: rewrite the query to replace the sub-structure p/c by p
   end while
8: //step 2: perform structural search on the rewritten query
   with new inverted lists
9: process the rewritten pattern of Q using any existing
   efficient structural join algorithm like Twig Stack, to
   get labels for output nodes.
10: remove newly created inverted lists
11: extract actual values with labels from corresponding
    tables, if the output node is a property ode; otherwise
    access the document to return sub trees.

Q1: //subject//book/title=“Network”//name
   In this query, //subject//book/title/name is a structural search.
   Aiming to find all matches in the document that
   satisfy this structural constraint; while the predicate
   title="Network" is a content search, which filters the structural search

XML query processing has been studied for over a
decade. In most XML query languages, e.g., XPath [4] and
XQuery [5], queries are expressed as twig patterns. Finding all occurrences of a twig pattern query in an XML
document is considered the core operation for XML query
processing. This process is also referred as twig pattern
query processing or twig pattern matching. Normally an
XML query is composed of structural search and content
search. Consider an XPath query Q1 that finds the subject
name of the book with the title of “Network”, issued to the
XML data in between structural search and content search on
e-library(fig.4). This attempt is proven efficient for structural
search without considering values. However, due to the
different characteristics between leaf value nodes and
internal non-value nodes in XML data, using inverted lists to
manage values and to process content search in the same way
as structural search will cause problems in managing
tremendous number of inverted lists and performing costly
structural join for content search.

Fig 4: e-library

Besides the inefficiency in content search, which
is caused by ignoring the semantic information of value
and non-value nodes, existing approaches may also
suffer from efficiency problems in structural search on
To look deeper into the semantics of non-value document
nodes, we can find that a non-value document node
may further correspond to an object or a property. Most
real life queries aim to find desired objects based on the
value predicates on their properties.

However, none of existing approaches takes
semantics of object and property into account when
they manage inverted list index and process queries.
The ignorance of such semantics would result in scanning
many useless labels in inverted lists during structural
search. In this paper, we propose a semantic approach for
twig pattern query processing. Motivated by solving the
problems caused by a lack of using semantics on object,
property and value in existing approaches, we propose
semantics-based relational tables incorporated with
inverted lists of tags to aid twig pattern query processing.

In particular, relational tables are used to store
values, while inverted lists are used to index internal
document nodes, including property nodes and object
nodes, but not values nodes. We design Twig Table
algorithm to perform content search and structural search
separately with the two kinds of indexes in twig pattern
matching. Content search is performed by table selection
before structural search. Because content search is always a
predicate between a property and a value, after performing
content search the size of the inverted list of the relevant
property node is reduced due to the selectivity of the
predicate, and the twig pattern query can be simplified by
removing value comparisons.

Matching a simplified twig pattern with reduced
inverted lists for several query nodes will reduce the
complexity of structural search, and thus improve the twig
pattern matching performance. Finally, the semantic tables
can help to extract actual values to answer the queries that
ask for property values or object details, which are not
efficient to achieve in other structural join based
algorithms. We also need to highlight that the relational
tables are constructed based on semantic information such
as the relationship among object, property and value.

The semantics of property is common for any XML
document, i.e., the parent node of each value must be the
property of that value. Based on this default semantic
information, we initially store each value with its property
in the corresponding property table. The bookstore
document with all nodes labeled by containment scheme
we propose three optimization techniques to change the
tables to be object based.

We will show that using object-based tables, a query
can be processed even more efficiently. In a word, with
more semantic information known, our approach is more
efficient to process twig pattern queries.

7. DATA MODEL AND PROCESSING

Normally an XML document is modeled as an
ordered tree, without considering ID references. The tree
structure of a bookstore document. In an XML tree, the
internal nodes represent the elements and attributes in the
document, and the leaf nodes represent the data values
which are either a text in an element or an attribute value.
Thus a node name is a tag label, an attribute name or a
value. Edges in an XML tree reflect element-sub element,
element attribute, element-value, and attribute-value pairs.

Two nodes connected by a tree edge are in parent-
child (PC) relationship, and the two nodes on the same
path are in ancestor-descendant (AD) relationship. The core
query pattern in most standard XML query languages
(e.g., XPath and XQuery) is also in a tree-like structure,
which is often referred as a twig pattern. In particular, an
XPath query is normally modeled as a twig pattern query,
and an XQuery query is normally modeled as several twig
patterns linked by joins. For example, the XPath query Q1
i.e., //subject//book/title="Network")/name,
can be represented as a twig pattern query.

In a twig pattern query, an edge can be either single-lined or double-lined, which constraints the two matched nodes in either a PC relationship or an AD relationship. Since a twig pattern normally models an XPath expression, we allow the leaf nodes of a twig pattern query to also be a range value comparison or even a conjunction/disjunction of several value comparisons, if the corresponding XPath expression contains such predicates. For example, the twig pattern representation of the

XPath query //book[price>20 and price<30]/title contains a conjunction of value comparison “>20 and <30” under the query node price.

The process to find all the occurrences of a twig pattern in an XML document is called twig pattern matching. A match of a twig pattern Q in a document tree T is identified by a mapping from the query nodes in Q to the document nodes in T, such that: (i) each query node either has the same string name as or is evaluated true based on the corresponding document node, depending on whether the query node is an element/attribute node or a value comparison.

8. SECURITY IN SOFTWARE

System security refers to various validations on data in form of checks and controls to avoid the system from failing. It is always important to ensure that only valid data is entered and only valid operations are performed on the system. The system employees two types of checks and controls:

Client side validation

Various client side validations are used to ensure on the client side that only valid data is entered. Client side validation saves server time and load to handle invalid data. Some checks imposed are:

- VBScript in used to ensure those required fields are filled with suitable data only. Maximum lengths of the fields of the forms are appropriately defined.
- Forms cannot be submitted without filling up the mandatory data so that manual mistakes of submitting empty fields that are mandatory can be sorted out at the client side to save the server time and load.
- Tab-indexes are set according to the need and taking into account the ease of user while working with the system.

Server side validation

Some checks cannot be applied at client side. Server side checks are necessary to save the system from failing and intimidating the user that some invalid operation has been performed or the performed operation is restricted. Some of the server side checks imposed is:

- Server side constraint has been imposed to check for the validity of primary key and foreign key. A primary key value cannot be duplicated. Any attempt to duplicate the primary value results into a message intimating the user about those values through the forms using foreign key can be updated only of the existing foreign key values.
- User is intimating through appropriate messages about the successful operations or exceptions occurring at server side.
- Various Access Control Mechanisms have been built so that one user may not agitate upon another. Access permissions to various types of users are controlled according to the organizational structure. Only permitted users can log on to the system and can have access according to their category. User- name, passwords and permissions are controlled the server side.
- Using server side validation, constraints on several restricted operations are imposed

9. CONCLUSION

In this paper, we evaluated the performance of our scheme in the experiments, compared not only to the previous wireless XML streaming methods but also conventional XML query processing methods supporting twig pattern matching. The conclusions of our study suggest that domain-specific knowledge improves the results. Ensuring that no information is lost over time is a problem that research areas such as interoperability, archiving and version management aim to make a contribution towards solving.

Temporal querying and version management are very challenging areas of research that pose a range of problems. Although this is reasonable query for one to have, expressing this query in conventional SQL would be quite challenging.

Performing similar kinds of queries on a collection of large XML documents on a regular basis would be inefficient at best. Our Project work focuses on how to minimize downloads of data from the wireless channel with shortest latency in wireless mobile environment.

10. FUTURE ENHANCEMENT

The enhancement which can be done is to extend the experimental methodology when the first setting is applied for the second task, to use additional sources of information as representation techniques, and to focus more on ways to integrate the research discoveries in a framework to be deployed to consumers. The greatest cost involved is with the temporal scan of possible file versions that is required for a query. Although a much wider variety of queries could have been possible, the queries that have been implemented provide non-XML accustomed users and XML proficient users with a way to query multi-versioned XML documents in a meaningful way.

REFERENCES