IOSR Journal of Computer Science (IOSR-JCE)
e-ISSN: 2278-0661, p-ISSN: 2278-8727
PP 29-34
www.iosrjournals.org

XML Based Query Engine

Mr. Ahmar Ali[1], Dr. M. M. Raghuwanshi[2],Mr .Santosh Kumar Sahu[3],

[1]*(PG Scholar, Department of Computer Science And Engineering, Rajiv Gandhi College of Engineering And Research, Nagpur,India)*

[2]*(Professor ,Department of Computer Science And Engineering, Rajiv Gandhi College of Engineering and Research, Nagpur,India)*

[3]*(Assistant Professor ,Department of Information Technology, Rajiv Gandhi College of Engineering And Research, Nagpur,India)*

ABSTRACT : *XML (Extensible Mark-up Language) has emerged as the leading textual language for representing and exchanging data on web. With the popularity of XML technology and Native XML database, efficiently indexing, and retrieving XML data in Native XML database become an important research topic.*

This project describes the description of advanced features of XML data which enables the retrieval and restricting of XML document in a simple and intuitive way. After the basic features of XML the focus is on Complex Queries whose expressive power subsumes some well known relational operations (e.g. union, difference, car titian products) but goes well beyond these relatively simple operations.

A single query engine can execute multiple queries at the same time and execution of a single query can be distributed across different servers. The distributed servers are totally symmetric and queries can be submitted to any server. The server conforms to XML and proposed XML-QL standards and all communication among servers follow these specifications. This makes sure that even the query engines from other vendors can participate in a distributed query execution.

***Keywords* -Inverted index, Multithereaded pipelined,Native XML, XML-QL,**

I. INTRODUCTION

Extensible Mark-up Language (XML) is a language that defines a set of rules for encoding documents in *a format that is both human readable and machine-readable. It is defined in theXML 1.0 Specification produced by the W3C, and several other related specifications, all gratis open standards. The design goals of XML emphasize simplicity, generality, and usability over the Internet[9].*

There are three important approaches to modelling semi structured SGML data. They are Traditional information retrieval, relational model and object-oriented approached by Sacks-Davis[1]. Wang Xiaoling[2] proposed a method based on genetic algorithm for XML document storage to seek the optimum mapping relational between XML Schema and relational database table. The XML schema will provide a means of using XML instances to define augmented DTDs. The Transformation adopts a database reverse engineering approach. Proposed by Joseph Fong, Francis Pang[6].

A. *SAX Parser:*

SAX (Simple API for XML) is an event-based sequential access parser API developed by the XML-DEV mailing list for XML documents. SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the Document Object Model (DOM). Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially[3].

B. ***Definition****:*

Unlike DOM, there is no *formal* specification for SAX. The Java implementation of SAX is considered to be normative. SAX processes documents state-dependently, in contrast to DOM which is used for state-independent processing of XML documents[3].

C. ***Benefits****:*
SAX parsers have some benefits over DOM-style parsers. A SAX parser only needs to report each parsing event as it happens, and normally discards almost all of that information once reported (it does, however,
keep some things, for example a list of all elements that have not been closed yet, in order to catch later errors such as end-tags in the wrong order).
Thus, the minimum memory required for a SAX parser is proportional to the maximum depth of the XML file (i.e., of the XML tree) and the maximum data involved in a single XML event (such as the name and attributes of a single start-tag, or the content of a processing instruction, etc.).

The compensating advantage, of course, is that once loaded any part of the document can be accessed in any order.Because of the event-driven nature of SAX, processing documents is generally far faster than DOM-style parsers, *so long as* the processing can be done in a start-to-end pass. Many tasks, such as indexing, conversion to other formats, very simple formatting, and the like, can be done that way. Other tasks, such as sorting, rearranging sections, getting from a link to its target, looking up information on one element to help process a later one, and the like, require accessing the document structure in complex orders and will be much faster with DOM than with multiple SAX passes[3].

In the current situation if we want to retrieve data from XML file, first we have to convert it into some standard relational database format. This paper proposes a keyword-based query solution for Native XML database. Each XML document can be presented as a rooted, ordered labelled tree. An inverted index for XML documents in Native XML database is constructed firstly, and inverted index s taken as the unique identify of the elements in XML documents. Then a query interface is designed on the basis of inverted index; query interface accepts a set of keywords from user, and then automatically generates executable FLWOR expression; finally FLWOR expression is sent to XQuery engine which is the important component of Native XML database system. The SAX parser is use to parse the XML file. SAX provides a mechanism for reading data from an XML document and operates on each piece of XML document sequentially. In early XML Based Query Engine *user provide a relational database file of his database to the data analyst, In XML Query Engine user get the GUI options to create the Complex Query and also able to provide the output format, the column name which he wants as an output.* The optimization can also perform on query engine.

D. ***XML processing with SAX parser****:*

A parser that implements SAX (i.e., *a SAX Parser*) functions as a stream parser, with an event-driven API. The user defines a number of callback methods that will be called when events occur during parsing[3]. The SAX events include (among others):

1) XML Text nodes
2) XML Element Starts and Ends
3) XML Processing Instructions
4) XML Comments

Some events correspond to XML objects that are easily returned all at once, such as comments. However, XML *elements* can contain many other XML objects, and so SAX represents them as does XML itself: by one event at the beginning, and another at the end. Properly speaking, the SAX interface does not deal in *elements*, but in *events* that largely correspond to *tags*. SAX parsing is unidirectional; previously parsed data cannot be re-read without starting the parsing operation again.

There are many SAX-like implementations in existence. In practice, details vary, but the overall model is the same. For example, XML attributes are typically provided as extreme name and value arguments passed to element events, but can also be provided as separate events, or via a hash or similar collection of all the attributes. For another, some implementations provide "Init" and "Fin" callbacks for the very start and end of parsing; others don't. The exact names for given event types also vary slightly between implementations [3].

A Relational database is a means of storing information in such a way that information can be retrieved from it. In simplest terms, a relational database is one that presents information intables with rows and columns. A table is referred to as a relation in the sense that it is a collection of objects of the same type (rows). Data in a table can be related according to common keys or concepts, and the ability to retrieve related data from a table is the basis for the term relational database. Using Relational database to save and manage XML data can bring many advantages for different users. Nowadays relational database is the mainstream database, changing XML data into relational data can not only reserve the specialty of easy to express and independent of platform in the aspect of XML at data application, moreover may fully using the mature data management service of relational database(effective memory, highly effective inquiry, concurrent control, data restore and so on), so making up the obvious shortcoming of XML technology in the aspect of searches, modification, achieving the goal of effectively manages and protects the XML data[4].

II. METHODOLOGY

1) *WORK FLOW/ARCHITECTURE OF DISTRIBUTED QUERYING*

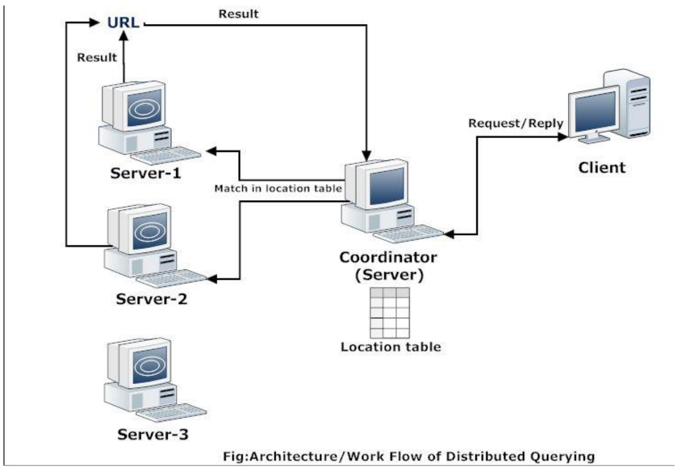

Fig. 1. Architecture of Distributed Querying

The Query Engine follows client-server architecture. The server waits on a particular port for requests from clients. The clients and servers communicate using sockets. The moment a request comes to a server, it spawns a separate thread to service that request. A single server can execute multiple queries concurrently and is always available to accept more requests if there are sufficient resources on the machine it is running on. If the servers are symmetric the server that first receives the query acts as a query coordinator. The client server architecture allows the query coordinator to act like a client and send sub queries to the other servers. The protocol is that whenever a server receives a query, it does not send the result of the query directly but returns a URL where it is going to place the results. The coordinator parses the query and looks at the names of XML documents that are needed in this query. [9]

Case 1. If none of the needed documents are in the location table, this server goes ahead and fetches these documents and executes the query itself.

Case 2. If there are some documents in the location table and this itself is the server for those documents then it behaves like in case 1.

Case 3. If there are some documents in the location table and the preferred server for those documents is not the coordinating server then the coordinator pushes the pattern matching for each of the documents to their respective servers.

The coordinator does not push the CONSTRUCT clause to any of the servers. All the final data comes back to the coordinator which then assembles the resulting XML document. The coordinator does not receive any intermediate results--it only receives the final results. The intermediate results are directly transferred among servers. Due to compatibility reasons, servers communicate among each other only using the standard XML and XML-QL and all documents, including intermediate results, are accessed only through URL's. The intermediate results are recast into XML format and shipped. The receiving server parses the document and continues its partof the query. Our query engine is able to handle the cases when multiple documents in a single inClause belong to different hosts and when documents of different inClauses belong to different hosts. [9] To achieve the distributed query system client-server architecture the complex queries has to be constructed in the query engine. The complex queries contain the operations like union, difference; joins, types of unions etc are to be implemented. These complex queries works similar in XML as it works in the relational database. [9]

2) ***Workflow of XML Query Engine****:*

First the query engine follows client-server architecture. The server waits on a particular port for requests from clients. The clients and servers communicate using sockets. The moment a request comes to a server, it spawns a separate thread to service that request. A single server can execute multiple queries concurrently and is always available to accept more requests if there are sufficient resources on the machine it is running on. If the servers are symmetric the server that first receives the query acts as a query coordinator. The client server architecture allows the query coordinator to act like a client and send sub queries to the other servers. The protocol is that whenever a server receives a query, it does not send the result of the query directly but returns a URL where it is going to place the results. The coordinator parses the query and looks at the names of XML documents that are needed in this query. [9]

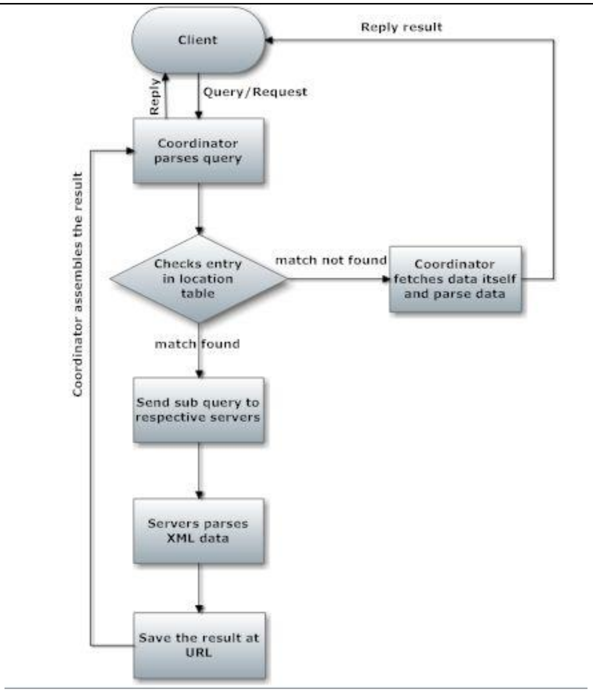
Client
Reply result
Reply
Query/Request
Coordinator parses query
Coordinator assembles the result
Checks entry in location table
match not found
Coordinator fetches data itself and parse data
match found
Send sub query to respective servers
Servers parses XML data
Save the result at URL

Fig. 2. Workflow of XML Query Engine.

Than the XML file is inputted to the XML Query Engine as shown in Fig. 2. The XML file is parsed by SAX parser after that the user selects the output format i.e. the column he wants as output. User also selects the column name and the values on which he applies conditions. User than add the query and get the result as XML file.

III. CONCLUSIONS

A "XML Query Engine" in which user provide a XML file and apply the XML Query Engine to perform the data analysis. In XML Query Engine user get the GUI options to create the Query and also able to provide the output format, the column name which he wants as an output. XML Query Engine performs the data analysis on relational database file after converting it into database schema. It reduces the query response time.

This proposed approach for distributed query engine works on xml file directly and performs the query. The client send the request to the coordinator , the coordinator searches the location table and send sub queries to the respective servers. The server parses the xml file and saves the result on URL. The coordinator assembles the result and replies result to the client.

REFERENCES

Journal Papers:

[1]. *Sacks-Davis, R., Arnold-Moore, T., Zobel, J.: Database Systems for Structured Documents. IEICE Transactions on Information and Systems, Vol. E78- D, No.11(1995)pp.1335-1341.*
[2]. Wang Xiaoling, Lan Jinfeng, Dong Yisheng, Basing on GA to Store XML data in RDBMS[j]. Journal ofComputer Research and Development,2003,40(7):111- 1116.
[3]. ZHOU Ao-Ying, XU Zheng-Chuan, GUO Zhi-Mao, etat.Adaptation of XML Storage Schema in VXMLR[J]. Chinese Journal Of Computes, 2004, Vol.27 No.4: 433-441.
[4]. Xie Yi-wu; Wang Chen-Yang; Cao Zhi-Ying; Chen Yan : Research on Store XML Data in Relational Database Based on XML Schema. Network and Parallel Computing Workshop, 2007. NPC Workshops. IFIP International Conference on. Digital Object Identifier: 10.1109/NPC.2007.126. Publication Year : 2007, Page(s):1001-1005.

[5]. Fong, J.; Pang, F.; Bloor, C.: Converting Relational Database Into XML Document. Database and Expert Systems Application, 2001 Proceedings: 12th International Workshop on Digital Object Identifier: 10.1109/DEXA.2001.953042. Publication Year : 2001, Page(s):61-65.

[6]. Xiangguo Zhoa; Junchang Xin; Ende Zhang: XMLFunctional Dependency and Schema Normalization. Hybrid Intelligent System, 2009. HIS'09. Ninth International Conference on. Digital Object Identifier: 10.1109/HIS.2009.276. Publication Year : 2009, Page(s):307-312.

[7]. Paramjit Oberoi and Vishal Kathuria "A Distributed Query Engine for XML-QL"University of wisconsin and Madison, {param,vishal}@cs.wisc.edu.

Proceedings Papers:

[1] Hiroto Kurita†, Kenji Hatano‡, Jun Miyazaki†, and Shunsuke Uemura†"Efficient Query Processing for Large XML Data in Distributed Environments"Graduate School of Information Science, Nara Institute of Science and Technology Keihanna Science City, Ikoma, Nara 630 0192, Japan ‡ Faculty of Culture and Information Science, Doshisha University 1-3 Tatara-Miyakodani, Kyotanabe, Kyoto 610-0394, Japan 21st International Conference on Advanced Networking and Applications(AINA'07) 0-7695-2846-5/07 $20.00 © 2007 IEEE

[2] Guilherme Figueiredo1,3, Vanessa Braganholo2, Marta Mattoso31 Brazilian Development Bank (BNDES), Brazil2 IC, Fluminense Federal University (UFF), Brazil 3 COPPE, Federal University of Rio de Janeiro (UFRJ), Brazil"Processing Queries over Distributed XML Databases" Journal of Information and Data Management, Vol. 1, No. 3, October 2010, Pages 455–470