Designing of an XPath Engine for P2P XML Store

Jovic, Darko; and Milutinovic, Veljko

Abstract— *In the introductory part, this paper defines the general environment for this research and defines the terms of interest. Then we define the research problem: How to perform distributed XPath query execution in a P2P environment. Existing solutions to the problem are briefly surveyed, and their drawbacks are underlined; each surveyed piece of research is analyzed according to the same template. Then, the essence of the proposed solution is presented: implementation of an XPath engine which works in a P2P environment and **performs distributed query execution. The conclusion is from the point of view of performance/complexity ratio.***

1. INTRODUCTION

XPATH is a language for addressing parts of an XML document, designed to be used by both XSLT and XPointer [1][4]. XPath expressions are used to query XML data, express transformations and reference elements in remote documents.

XPath is specification language designed by the World Wide Web Consortium (W3C) [6]. It is an expression language for addressing parts of an XML document, used by various XML technologies such as XSL Transformation (XSLT) and XML Pointer Language (XPointer). As the name implies XPath uses path notation for navigating through the hierarchical structure of an XML document. Like Document Object Model (DOM), XPath views an XML document as a tree structure consisting of nodes of different type, each representing the entities of a document.

2. PROBLEM STATEMENT

Our problem was to design an XPath engine which performs distributed XPath query execution on a distributed data storage system. System stores XML documents in a P2P community (Figure 1) by spreading parts of the document among peers in the community. It is based on JxTa P2P framework, implementation of distributed hash tables and implementation of Document object model (DOM) level 2 interfaces [2][3].

We can use XPath engine which works with generic DOM and use it on top of P2P DOM layer. This way we get full XPath functionality, but query execution is performed locally. All required node sets are pulled from the community and nodes matched against query to get the result. Getting node sets from the community in case of large XML documents can cause increased communication among local peer and other peers which contain node sets for specific documents. High communication traffic can be avoided by distributing XPath query execution.
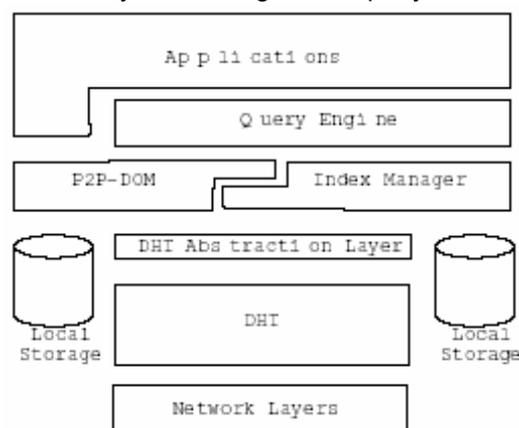

Figure 1: Peer-to-Peer XML Storage System

3. EXISTING SOLUTIONS

XML Store [7] is a distributed and scalable storage system for XML documents that builds on the Chord P2P protocol developed at MIT. It has adaptation of XPath which supports very reduced set of XPath 1.0 functionality. Only one axis is supported – *child* axis, and only one kind of predicate – *position()* operation. What makes a challenge is to build an XPath engine which works in P2P environment and supports full XPath 1.0 specification.

4. PROPOSED SOLUTION

High communication traffic and memory consumption can be avoided by implementing an XPath engine, which performs distributed XPath query execution

Since there are several open source engines for Java, we decided not to write the XPath engine from scratch, but to modify the existing one which best fits our needs and which will take less time to adapt to work in a P2P environment using P2P DOM layer. Since P2P DOM layer implements DOM interfaces we should choose

engine which works with generic DOM interfaces, and not with specific implementations of DOM.

4.1 Existing XPath Engines

Most of these open source XPath engines are distributed as part of XSLT processor. We will give a short description of few of them with the focus on what kind of XML object model they support.

SAXON

SAXON package is a collection of tools for processing XML documents. It includes:

- XSLT processor, which implements the Version 1.0 XSLT and XPath Recommendations from the World Wide Web Consortium, with a number of powerful extensions.

- A Java library, which supports a similar processing model to XSL, but allows full programming capability, which you need if you want to perform complex processing of the data or to access external services such as a relational database

- A slightly improved version of the Ælfred parser from Microstar. (But you can use SAXON with any SAX-compliant XML parser if you prefer).

Saxon requires a custom DOM that has been annotated with the information it needs. You can't just pass in a Xerces DOM or a Crimson DOM, but you must use JAXP to build Saxon Document object with Saxon's implementation of DocumentBuilderFactory interface.

Xalan – J

The Xalan-J XSLT processor from the Apache XML Project also includes an XPath API that's useful for navigation in DOM programs. Xalan-Java (named after a rare musical instrument) fully implements XSL Transformations (XSLT) Version 1.0 and the XML Path Language (XPath) Version 1.0.

Xalan-J works with generic DOM implementations, but it has a little complicated architecture, so it is not ideal for modifications.

Jaxen

The Jaxen Java XPath Engine is an open source cross-API (DOM, JDOM, dom4j, and ElectricXML) XPath library for Java. Whereas DOM3 XPath [5] attempts to be a cross-implementation, cross-language XPath API for the Document Object Model alone, Jaxen attempts to be a cross-model API for its own XPath engine. It is a Java class library that can operate on various XML object models using a standard engine rather than an API that can be offered by many different engines for one model. It allows you to pass DOM, JDOM, dom4j, and ElectricXML objects directly to XPath functions like position() and translate(). Furthermore, whereas DOM 3 XPath offers fairly rudimentary interfaces for evaluating XPath expressions in a particular document against a context node, Jaxen is a more complete object model for XPath expressions.

Jaxen supports different XML object models through different implementations of org.jaxen.Navigator interface. By implementing this interface we can use Jaxen on any XML object model. But still it is not enough to adapt it to work in a P2P environment. Jaxen packages have clear hierarchy so it makes it less difficult to apply modifications to its classes. For this reasons we chose this XPath engine as a basis for our XPath engine which performs distributed XPath query processing.

4.2 Distributed Processing of XPath Queries

The primary syntactic construct in XPath is the location path, which is a sequence of location steps separated by a slash (/). Each step in turn selects a set of nodes relative to the context node and each node in that set is used as a context node for the following step. The syntax of a location step consists of an axis, a node test and zero or more predicates.

For example:

/child::weather/child::day[attribute::date='2001-12-12']/child::readings/child::reading

In a P2P XML storage XML document can be split into several node sets, which are stored on different peers. P2P DOM layer implements DOM interfaces so XPath engine can use these interfaces to evaluate expressions against XML documents, without needing to know underlying architecture of the P2P DOM layer. Node sets which contain nodes requested by the XPath engine will be pulled from the community to the local peer, so evaluation of the XPath expression can be performed. As we can see we can use any XPath engine which works with generic DOM level 2 interfaces to evaluate XPath expressions against some XML document stored in the P2P community. But this approach has certain drawbacks like memory consumption and time consumption. DOM representation of the XML document takes quite a memory, especially if the document is large. For every node in the document a corresponding object must be created in memory. Each object has links – keys to its owning document, parent node, child nodes, siblings, and also has namespace URI, name, etc. For evaluating XPath expression entire document tree needs to be in memory, which can sometimes lead to lack of memory problems if the document is extremely large. In order to evaluate XPath expression XPath engine must visit larger part of the document tree and get most of the node sets for the document from the community. Gathering these node sets from the community and creating DOM constructs takes some time, so the performance of evaluating XPath expression in this way would not be acceptable in everyday usage.

To overcome time and memory consumption during XPath query processing, we need to distribute the query processing so that peers who owns the node sets for one document take place in evaluation process. Each peer should evaluate part of the XPath query and return the result to the requesting peer, thus contributing in a chain of XPath query evaluation process.

XML Store [7] is a distributed and scalable storage system for XML documents that builds on the Chord P2P protocol developed at MIT. It has adaptation of XPath which supports very reduced set of XPath 1.0 functionality. Only one axis is supported – child axis, and only one kind of predicate – position() operation. What makes a challenge is to build an XPath engine which works in P2P environment and supports full XPath 1.0 specification.

4.2.1 XPath Query Evaluation and Migration

Query evaluation starts from the local peer by evaluating the first location step of the expression. The XPath engine iterates over nodes along the specified axis and when next node along the axis is not stored locally then query is migrated to the peer who owns the next node. When we say that the node is placed locally, it is assumed that the node is located in the persistent XML storage of the peer or in the cache buffer. Expression migration includes several steps:

- Extracting part of the XPath expression which is not processed and converting it to new expression suitable for migration. This part of the expression includes current location step which is not fully processed, and all following location steps in the expression.

- Determining next node along the axis which is not stored at the local peer. This node is new context node for the converted expression;

- Extracting current XPath context from the engine. Context includes namespace mappings and variables;

- Creation of message which contains above mentioned data: converted XPath expression, reference of the new context node and XPath context. This message is sent to the peer which owns the referenced node.

After message is sent to destination peer, current query processing is stopped until the result is received. Remote peer receives the message, recreates XPath context from the data contained in the message, evaluates expression against selected context node and returns the result. After result is received from the remote peer, nodes which are processed before migrating expression are joined to form the unique set of nodes, which form the result of evaluating the expression for the current context node. As we said before, nodes selected by one location step are context nodes for the following location step in the expression. During the expression evaluation we can get to some location step for which we can't perform full evaluation because not all nodes are stored locally. At this moment we have context nodes which are selected from the previous location step. We iterate through each context node and try to evaluate the result of the current location step. If migration is necessary then not only current location step is evaluated at remote peers, but the entire rest of the expression. Evaluating the rest of the location steps for current context node is finished and we can go to next context node.

Peer who receives the message for evaluating XPath expression, during evaluation process can migrate the evaluation process to other peers, thus forming the chain of XPath evaluation process. Each peer waits the result from the peer to which message was sent, and upon receiving an answer it forms the final result and sends it to the requesting peer.

4.2.2 XPath Expression Conversion

In order to successfully migrate the XPath expression conversion must be performed. The reason for this is that the context node must be changed during the migration. When expression needs to be migrated to another peer, current context node is the node which is stored at the local peer. This node does not need to be stored at the remote peer, and because of that it can not be context node for the expression which is sent to remote peer. New context node is the node which needs to be visited next and is located at the remote peer. Let's take for example child axis. If we have location step child::item and the current context node is Items node. If for example third and fourth node are located at different peer then the other item nodes, then when evaluation gets to the point where third item node needs to be visited expression must be migrated with the third item node as new context node. With that in mind we can not get appropriate results if we do not change axis. If we use the same expression *child::item* on the third item node we will not visit the fourth item node because it is the sibling of the third item node and not the child node. For that reason expression needs to be converted to *following-sibling::item*. Also we must include the new context node and thus get the final form of the expression: *self::item | following-sibling::item*, where '|' is union operator.

Here is the list of each axis and corresponding conversion (rest_of_expr represents current locations step without axis and following location steps in the expression):

• **Child** axis is converted to **self::*rest_of_expr* | following-sibling::*rest_of_expr***

• **Parent** axis – **self::*node_test***

- **Descendant** axis – **following-sibling::*rest_of_expr* | descendant-or-self::*rest_of_expr***

- **Descendant-or-self** axis - **following-sibling::*rest_of_expr* | descendant-or-self::*rest_of_expr***

- **Ancestor** axis – **ancestor-or-self::*rest_of_expr***

- **Ancestor-or-self** axis – **no conversion**

- **Following** axis – **self::*rest_of_expr* | following::*rest_of_expr***

- **Following-sibling** axis – **self::*rest_of_expr* | following-sibling::*rest_of_expr***

- **Preceding** axis - **self::*rest_of_expr* | preceding::*rest_of_expr***

- **Preceding-sibling** axis - **self::*rest_of_expr* | preceding-sibling::*rest_of_expr***

- **Self** axis – **no conversion**.

4.4 Modifications of P2P DOM Layer and Jaxen XPath Engine

In order to support distributed XPath query processing, in our system (Figure 1), P2P DOM layer needed to go through certain changes and modifications.

Link between P2P DOM layer and XPath engine needed to be created so that queries could be passed from P2P DOM layer to XPath engine and enable XPath engine to get necessary data from P2P DOM layer.

New type of message needed to be supported. This message is used to transfer necessary XPath data to remote peer. Also, response message needed to be expanded to support result of the XPath query evaluation.

Constructs for supporting XPath context–namespaces, variables and functions, so that context data can be sent to remote peer and there reconstructed so that query can be processed.

Jaxen was modified to enable communication with lower P2P DOM layer, and to migrate queries to remote peers. Algorithm of query evaluation was adapted to inspect if some node is stored locally, and if this is not the case to request migration of part of the XPath expression.

5. CONCLUSION

We showed that it is possible to adapt existing XPath engine to process XPath expression in a distributed manner in a P2P environment. Expression is evaluated locally until there is condition for that. The condition is that nodes are stored at the local peer, either in the cache buffer, or in the persistent PDOM storage. If this condition is not met, unevaluated part of the expression is migrated to remote peer, where its

evaluation continues. There is one exception to this rule, that location step does not have a predicate which consist entirely of position operation. In this case the location step is evaluated locally.

The modified XPath engine was briefly tested with common XPath expressions.

REFERENCES

[1] Simpson, E., John, "XPath and XPointer," *O'Reilly*, August 2002
[2] Knežević, P., Risse, T., Neuhold, E., J., Fankhauser, P., "Designing a P2P XML Database – Applications and Open Issues"
[3] Jović, D., "P2P Store for Static P2P Communities, Sofware Architecture Document"
[4] "XML Path Language (XPath)," Version 1.0, *W3C Recommendation*, 16. Nov. 1999., http://www.w3.org/TR/1999/REC-xpath-19991116
[5] "Document Object Model (DOM) Level 3 XPath Specification," Version 1.0, *W3C Working Group Not*, 26. Feb. 2004., http://www.w3.org/TR/2004/NOTE-DOM-Level-3-XPath-20040226
[6] World Wide Web Consortium Web Site, http://www.w3.org
[7] Thorn, T., Baumann, A., Fennestad, M., "A distributed, value-oriented XML Store", Master's Thesis, *IT University of Copenhgen,* August 2002, http://www.itu.dk/xmlstore/XMLStore.pdf