Job Finding System by Using XML Document Filtering: XFilter

Phyu Thin Khaing, Khin Sandar
University of Computer Studies (Mawlamyine)
phyuthinkhaing@gmail.com

Abstract

The exchange of data, information and knowledge becomes a key issue over the Internet. XML (eXtensible Markup Language) also plays an important role as a de-facto standard online data exchange mechanism. In most web applications, XML data filtering methods are applied. The various filtering mechanisms have been developed for XML data such as XFilter, YFilter, AFilter and so on. This paper describes filtering of XML document. This system applies XML data filter called XFilter that provides the matching of XML document to large numbers of user profiles and delivering relevant data to the interesting users. XFilter intends to provide approximate information and queries which are matched with the incoming XML documents. Firstly, this paper describes how to parse the XML document using SAX (Simple API for XML) Parser. Secondly, how to decompose XPath queries in the user profile as path node using XPath Parser is described. Finally, the matching step will be carried out inside Filter Engine. After that, we produce the relevant information for providing the user. In this paper, we apply job information by matching the job XML Document and the user desired XML format with XFilter method.

1. Introduction

On the Internet, people can easily navigate a lot of web sites, but getting their desired Information relies mainly on user's input keyword. For the users, it becomes increasingly difficult and is time-consuming to find the relevant information of job they are looking for. The job finding system was developed to assist the users. Many publish-subscribe applications rely upon user profiles in order to effectively target the right information to the right people. In most applications on the Internet [5] (e.g., distribution of entertainment, stock quotes, electronic news and articles delivery, etc.,), the number of users for such systems can easily grow into millions. A key challenge in such environments is to efficiently and quickly look for the huge set of user profiles in the documents to match those for which the document is relevant.

Filtering systems have traditionally been developed using Information Retrieval techniques based on both the Boolean and the "bag-of-words" models [3]. In this paper, we apply document filtering system called XFilter which has focused on the efficient evaluation of path expressions over streaming XML data, using indexed Finite State Machine (FSM) to allow many structure-based queries to be processed simultaneously. FSMs are a natural and effective way to represent and process path expressions. Elements of a path expression are mapped to states. A transition from an active state is fired when an element is found in the document that matches that transition. If an accepting state is reached, then the document is said to satisfy the query. In XFilter, user interests are represented as queries using XPath language [1]. For the user queries, there are many XML query languages: XPath, Xquery, ApproXQL, XQL and so on. This paper uses XPath, which include parent-child (/) and ancestor-descendant (//) relationships.

The main part of our method is XPath parser, XML (SAX) parser and Filter engine. XPath parser decomposes each XPath query in the user profile. On the other hand, XML document is parsed by SAX parser. And then, the matching step will be carried out inside Filter engine. XFilter engine uses a sophisticated index structure at a modified FSM. After that, the matching document must be sent to the appropriate users.

The rest of the paper is organized as follows. Section 2 summarizes the related work. In section 3 we introduce some background theory. Section 4 presents implementation of system design and finding job system. Finally we conclude the paper in section 5.

2. Related Work

Y. Diao [3] gave a filtering engine called YFilter, which filters streaming XML documents according to XQuery or XPath queries that involve both path expressions and predicates. P. Antonellis [2] showed an efficient technique for matching user profiles that is based on efficient technique for matching user profiles that is based on the use of holistic twig-matching. This algorithm is able to handle order matching of user profiles, while its main positive aspect is the envisaging of a representation based on priifer sequences that permits the effective investigation of node relationships. K. S. Candan [7] presented AFilter, an adaptable and thus scalable, path expression filtering approach. AFitler can exploit prefix commonalities in the set of filter expressions using a loosely-coupled prefix caching

mechanism as opposed to tightly-coupled active state representation of alternative approaches. X. Gong [5] presented XML data filtering which uses Bloom filters representing path queries. J. Kwon [6] filtering system called FiST that transforms twig patterns expressed in XPath and XML documents into sequences using Priifer's method.

In this paper, we employed XFilter that provides the matching of XML document to large numbers of user profiles and delivering relevant data to the interesting users for our system implementation.

3. Background Theory

3.1 XFilter (XML Filtering Mechanism)

XFilter mechanism consists of the following components.

- XPath parser for user profiles
- Event-based parser for XML documents
- Filter engine that performs matching operations between profile and XML documents

XPath Parser: The XPath parser takes queries written in XPath, parses them, sends, the parsed profiles to the filtering engine.

Event-based XML Parser: When an XML document arrives at the system; it is run through the XML parser. [4]This system uses a parser based on the SAX interface, which is a standard interface for event-based XML parsing. A "start element" event carries information such as the name of the element, its attributes, etc. A "characters" event reports a string that is not inside any XML tag. An "end element" event corresponds to an earlier "start elements" event by specifying the element name and marks the close of the element in the documents.

Filtering Engine: At the heart of the system, the filtering engine takes the parsed profiles from the XPath parser and converts them to an internal representation. It also receives and reacts to events from the XML parser. These events call back the corresponding handlers implemented by the filtering engine, which perform the matching of documents against profiles.

3.1.1 Internal Profile Representation

Each XPath query is decomposed into a set of *path nodes* by the XPath parser. These path nodes represent the element nodes in the query and serve as the states of the FSM for the query [1]. Path nodes are not generated for wildcard ("*") nodes. A path node contains the following information:

QueyId: A unique identifier for the path expression to which the path node belongs (generated by the XPath Parser).

Position: A sequence number, representing the location of this path node in the order of the path nodes for the query [8].

RelativePos: An integer that describes the distance in *document levels* between this path node and the previous (in terms of position) path node. This value is set to 0 for the first node if it does not contain a descendant ("//") operator. A node that is separated from the previous one by such an operator is flagged with a special RelativePos value of -1. Otherwise, the RelativePos value of a node is set to 1 plus the number of wildcard nodes between it and its predecessor node [4].

Level: An integer that represents the level in the XML document at which this path node should be checked. If the node is the first node of the query and it specifies an absolute distance from the root (i.e., it is either applied to the root node or is a fixed number of wildcard nodes away from the root node), then the level for that node is set to 1 plus its distance from the root. If the RelativePos value of the node is -1, then its level value is also initialized to -1. Otherwise, the level value is set to 0.

NextPathNodeSet: Each path node also contains pointer(s) to the next path nodes of the query to be evaluated.

3.1.2 Execution Algorithm

When a document arrives at the Filter Engine, it is run through an XML Parser which then drives the process of checking for matching profiles in the Index.

Start Element Handler: A start element event calls the handler, looks up the element name in the Query Index and examines all the nodes in the candidate list for that entry [4]. For each node, it performs a level check. The purpose of the level check is to make sure that the element appears in the document at a level that matches the level expected by the query. If the path node contains a non-negative level value, then the two levels must be identical in order for the check to succeed [1]. Otherwise, the level for the node is unrestricted, so the check succeeds regardless of the element level.

If the check succeeds, then the node passes. If this is the final path node of the query (i.e., its final state) then the document is deemed to match the query. Otherwise, if it is not the final node, then the query is moved into its next state. This is done by *copying* the next node for the query from its wait list to its corresponding candidate list (note that a copy of the promoted node remains in the wait list) [4]. If the *RelativePos* value of the copied node is not -1, its

level value is also updated using the current level or depth of the XML document and its *RelativePos* values of the copied node to do future level checks correctly.

End Element Handler: When an end element tag is encountered, the path nodes promoted when the corresponding start element tag was encountered are deleted from the candidate lists in order to restore those lists to the state before reading this element.

Element Characters Handlers: The handler is called when the data associated with an element is encountered. The data is passed in to the handler as a parameter. It works similarly to the Start Element Handler except that it performs a content filter check rather than an attribute filter check [1]. That is, it evaluates any filters that reference the element content. Like the Start Element Handler, this handler can also cause the query to move to its next state.

4. Implementation of System Design

In this section we describe the processes to implement of system design as shown in Figure 1.

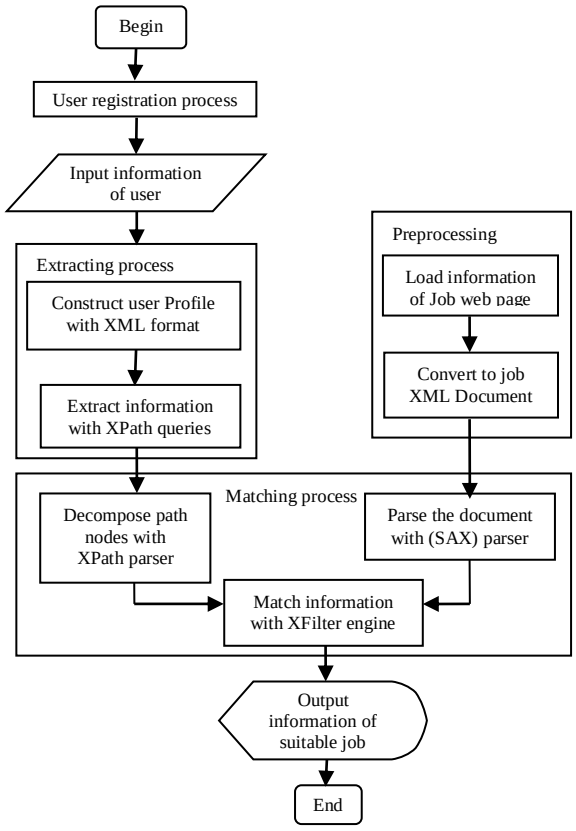

Figure 1: System design of XFilter

System design has three phases process.

Phase I: Preprocessing Phase
Step 1: Load information of job web page
Step 2: Convert XML document from HTML document
Step 3: Parse XML document with SAX parser

Phase II: Extracting Phase
Step 1: Register of user
Step 2: Take input information of user
Step 3: Construct user profile with XML format
Step 4: Extract information from user profiles by using XPath Queries
Step 5: Decompose as path nodes with XPath Parser

Phase III: Matching Phase
Step 1: Match information with XFilter engine
Step 2: Send relevant job information to user

4.1 Implementation of Job Finding System

4.1.1 Job Data for User

For our system, we intend to use job data for user. User information contains Category, Job Type, Job Title and Skill. This data is shown in Table 1.

Table 1: Example of job data for user

JobType	JobTitle	Skill
Category - Banking		
FullTime Project Contract	-Job Opening Banking Financial -Accounting Manager -Finance Head	-Accountancy -Banking
Category - Software		
FullTime Project Contract	-Software Developer -Software Programmer -Sybase Developer	-JavaScript -Java and Microsoft.Net
Category - Hardware		
FullTime Project Contract	-Network Support Specialist -Service Desk Engineer -Technical Support Engineer	-Window 98 and Hardware
Category - Design		
FullTime Project Contract	-User Interface Designer -Engineering Analyst -Oracle Database Administrator	-UI models -Web Products Planner
etc		

4.1.2 User Desired Information Format Converts to XML Format

User desired information format converts to XML format for matching job XML web page with XFilter engine and send relevant job to the user.

```
<? Xml version="1.0">
<Category>Software
        <JobType>Full Time</JobType>
        <JobTitle>Software Programmers
        </JobTitle>
        <Skills>SQL and MySQL, Database
        </Skills>
</Category>
```

After that, XPath queries extract query from user desired information that convert XML format. For eg, (first query Q1: Category/JobType/JobTitle, Second query Q2: //Skills that extract from user desired information).

4.1.3 XFilter Operation

In XFilter Operation, Firstly, job web page HTML format convert to XML format for matching user desired information with XFilter engine and send relevant job to the user.

(a) Job XML Format

```
<? Xml version="1.0">
<Category>Software  <-- Start element event
        <JobType>Full Time, Contract-Perm</JobType>
        <JobTitle>Software Programmers</JobTitle>
        <Skills>SQL and MySQL, Database, Software Engineering, Working knowledge of C</Skills>
</Category>
```

(b) Path Node Decomposition

For given user desired information that consists of two XPath queries, the XPath parser decomposes them into individual path nodes, each of which contains information such as *QueryId* (Q1, Q2), *Position*, etc.

Q: (Position, RelativePos, Level)
Q1: Category/JobType/JobTitle
Q1 (1, 0, 1), Q1 (2, 1, 0), Q1 (3, 1, 0)
Q2: //Skills
Q2 (1,-1,-1)

(c) Query Index

Then the corresponding Query Index is constructed and initialized such that the path nodes, whose *Position* values are 1, are placed in the Candidate List and others are placed in the Wait List inside their corresponding elements nodes. The Figure 2 summarizes how the XFilter system works with job example.

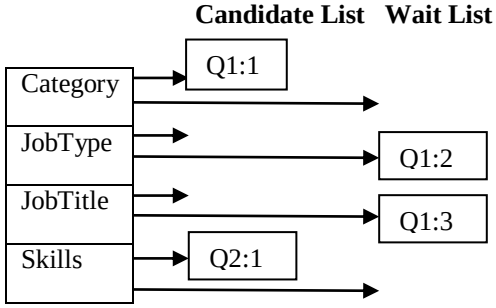

Figure 2: XFilter operations

(d) Matching Process

When a new XML job document arrives, it is parsed by the SAX parser that generates events whenever the structure of the Query Index [8]. For example, when the parser sees the <Category> node in job XML format, it generates the 'start-element' event. According to this event, the Filter Engine searches the Query Index for the key 'Category' and performs *level check and attribute filter check* on the path nodes in the Candidate List, which is only 'Q1-1' in Query Index. Since the level of the element node in the XML document, where the SAX event occurred, is same as the level value of the 'Q1-1' path node, the path node passes the check and the Filter Engine copies the next path node (Q1-2) and moves the copy from Wait List to the end of Candidate List of 'JobType' index. While it copies and moves the path node, it also updates the *level* value of the copied node using the *RelativePos* value of the copied node and the current level or depth of the XML document. In the example, the copied node of 'Q1-2 will have a new *level* value of 3, which is the result of 1 (*RelativePos* value) + 2(current level of XML document).

The SAX parser continues and then sees <JobType> node in the job XML format, finally generating the corresponding event. This process is repeated until the end of the document.

If the matching step succeeds, this system produces the relevant output (e.g., Software Programmers) message for job finding domain as shown in Figure 3.

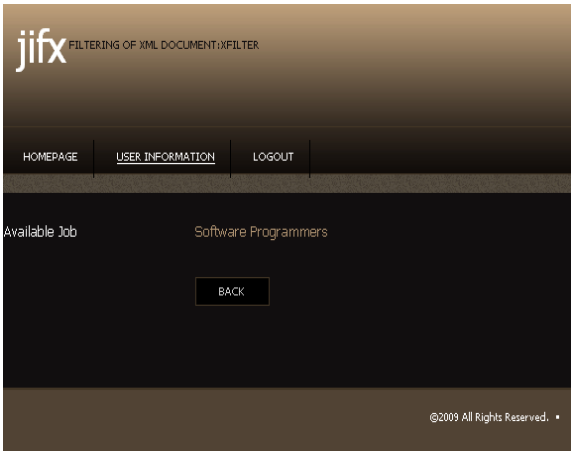

Figure 3: Relevant output message

After that, this system produces job web page of Software Programmers message as shown in Figure 4.

Figure 4: Job web page of software programmers

5. Conclusion

In this paper, we presented an XML document filtering system, called XFilter for finding job system. XFilter allows users to define their interests using the XPath query language. This paper developed indexing mechanisms and matching algorithm based on a modified Finite State Machine (FSM) approach that can quickly locate and evaluate relevant profiles. By converting XPath queries into a Finite State Machine representation, XFilter is able to (1) handle arbitrary regular expressions in queries, (2) efficiently check element ordering and evaluate filters in queries, and (3) cope with the semi-structured nature of XML documents.

This paper provides timely delivery of job information and filtering against user desired profile. It is to efficiently and quickly search the potentially huge set of user profiles to find those which the document is relevant. Our future work involves the integration of a variety of delivery mechanisms as investigated in our previous work and the delivery of partial documents.

References

[1] M. Altinel, M. J. Franklin, "Efficient Filtering of XML Documents for Selective Dissemination of Information", 26th VLDB Conference, Egypt, pages 53-64, Morgan Kaufmann Publishers Inc., 2000.

[2] P. Antonellis, C. Makris, "XFIS: an XML filtering system based on string representation and matching", Computer Engineering and Informatics Department, Patras University.

[3] Y. Diao, P. Fischer, M. Franklin, R. To, "YFilter: Efficient and Scalable Filtering of XML Documents", in Proceeding of the 18th International Conference, (ICDE 02) pages 341, IEEE Computer Society, 2002.

[4] Y. Diao, M. Altinel, M. J. Franklin, H. Zhang, "Path Sharing and Predicate Evaluation for High - Performance XML Filtering", ACM , 2003.

[5] X. Gong, W. Qian, Y.Yan, and A. Zhou, "Bloom Filter-based XML Packets Filtering for Millions of Path Queries", Fudan University, Sanghai, p.R.China.

[6] J. Kwon, P. Rao, B. Moon, S. Lee "FiST: Scalable XML Document Filtering by Sequencing Twig Patterns", VLDB conference, 2005.

[7] K.S. Candon, W.P. Hsing, S. Chen, J. tatemura and D. Agrawal, "AFilter: Adaptable XML Filtering with Prefix-Caching and Suffix-Clustering", ACM, 2006.

[8] H. Woo, P. Kang, Computer Sciences, "XML Filtering in Peer-to-peer Systems", University of Texas at Austin