ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 2, Issue 12, December 2013

Scalable Filtering and Matching of XML Documents in Publish/Subscribe Systems for Mobile Environment

Yi Yi Myint, Hninn Aye Thant

***Abstract*— The number of applications using XML data representation is growing rapidly, thus the process of XML filtering is becoming an essential need of publish/subscribe (pub/sub) system. The pub/sub model has gained acceptance as a solution for the loose coupling of systems in terms of asynchronous communication enabling Selective Dissemination of Information (SDI). SDI systems distribute the right information to the right users based upon their profiles. The inherent limitations of mobile devices necessitate information to be delivered to mobile clients to be highly personalized according to user profiles. In this paper, we address the issue of scalable filtering of XML documents for mobile environment. This paper proposes an approach that integrates pub/sub system and XML message filtering to deliver personalized information from XML resources to mobile clients. Finally, the proposed architecture describes an efficient indexing mechanism by enhancing XFilter algorithm based on a modified Finite State Machine (FSM) approach to achieve good scalability.**

***Index Terms*—FSM, indexing mechanism, SDI, XML**

I. INTRODUCTION

Extensible Markup Language (XML) filtering systems constitute a critical component of modern information-seeking applications. The advent of XML [3] as a de facto standard for information exchange and the development of query languages for XML data enable the development of more sophisticated filtering mechanisms. Emerging distributed information systems such as Web services, personalized content delivery, and event monitoring require increasingly flexible and adaptive infrastructures [4]. Recently, pub/sub systems are increasingly often used as a communication paradigm for loosely-coupled systems.

The pub/sub communication paradigm [10] can help to meet the challenge by providing efficient, scalable, many-to-many, push-based delivery of messages between the various parties in a system. In a pub/sub system, subscribers express their interests (profiles) in an event or a pattern of events, and are asynchronously notified when publishers produce them. The main purpose of an XML filtering system is to find all the user profiles that have a match with a specific XML document [8]. Data matching can be performed either at the source or at some centralized brokers. Several approaches for XML filtering have been reported in the literature, see Section 2 for details.

Expressing highly personalized profiles need a querying power just like SQL provides on relational databases. Since the queries will be executed on the documents fetched over the Internet, it is natural to expect the documents to be in XML, XML being the emerging standard for data exchange over the Internet. Then the user profiles need to be defined through an XML query language. XML-QL is a good candidate in this respect due to its expressive power as well as its elaborate mechanisms for specifying query results through the CONSTRUCT statement [12].

When such a system providing highly personalized services is deployed on the Internet, the performance becomes a critical issue since the number of users can easily grow dramatically. A key challenge is then to efficiently and quickly process the potentially huge set user profiles on XML resources. This boils down to developing efficient ways of processing XML-QL queries on XML documents.

The rest of the paper is organized as follows: Section 2 briefly summarizes the related work. In Section 3, the overall architecture of the system is described. The operation of the system that is how the query index is created, the operation of the finite state machine and the generation of the customized results are explained in Section 4. Section 5 gives the performance evaluation of the system. Finally Section 6 concludes the paper.

II. RELATED WORK

The filtering mechanism described in this paper is influenced by the XFilter system [1]. XFilter is designed and implemented for pushing XML documents to users according to their profiles expressed in XML Path Language (XPath). It provides efficient filtering of XML documents with the help of profile index structures in its filter engine. When a document matches a user's profile, the whole document is pushed to the user. This feature prevents XFilter to be used in mobile environments since the limited capacity

***Yi Yi Myint**, Faculty of Information and Communication Technology, University of Technology (Yatanarpon Cyber City), Pyin Oo Lwin, Mandalay Division, Myanmar.*

***Hninn Aye Thant**, Faculty of Information and Communication Technology, University of Technology (Yatanarpon Cyber City), Pyin Oo Lwin, Mandalay Division, Myanmar.*

of the mobile devices is not enough to handle the entire document. Furthermore, XFilter does not exploit the commonalities between the queries, i.e. it generates a FSM per query. This observation motivated us to develop mechanisms that use only a single FSM for the queries which have common element structure.

NiagaraCQ system [6] uses XML-QL to express user profiles. It provides measures of scalability through query groups and cashing techniques. However, its query grouping capability is based on execution plans which are completely different from our approach. The performance results do not make such architecture a possible candidate for mobile environments. YFilter [2] overcomes the disadvantage of XFilter by using nondeterministic finite automata (NFA) to emphasize prefix sharing. However, the ancestor/descendant relationship introduces more matching states, which may result in the number of active states increasing exponentially. Post processing is required for YFilter. To deal with queries with nested paths (complex queries), YFilter decomposes them into simple queries and matches them separately.

BFilter [7] conducts the XML message filtering and matching by leveraging branch points in both the XML document and user query. It evaluates user queries that use backward matching branch points to delay further matching processes until branch points match in the XML document and user query. In this way, XML message filtering can be performed more efficiently as the probability of mismatching is reduced. A number of experiments have been conducted and the results demonstrate that BFilter has better performance than YFilter for complex queries [11]. XFIS [9] proposes an efficient technique for matching user profiles that is based on the use of holistic twig-matching algorithms and was more effective, in terms of time and space complexities, in comparison with previous techniques. The proposed algorithm is able to handle order matching of user profiles, while its main positive aspect was the envisaging of a representation based on Prüfer sequences that permits the effective investigation of node relationships.

FoXtrot [5] is a system for filtering XML data that combines the strengths of automata for efficient filtering and distributed hash tables for building a fully distributed system. FoXtrot also describes different methods for evaluating value-based predicates. The performance evaluation demonstrates that it can index millions of user queries, achieving a high indexing and filtering throughput.

III. Overall Architecture of the System

This system proposes the architecture for mobile network to deliver highly personalized information from XML resources to mobile clients. The overall architecture of the system is depicted in Fig. 1.

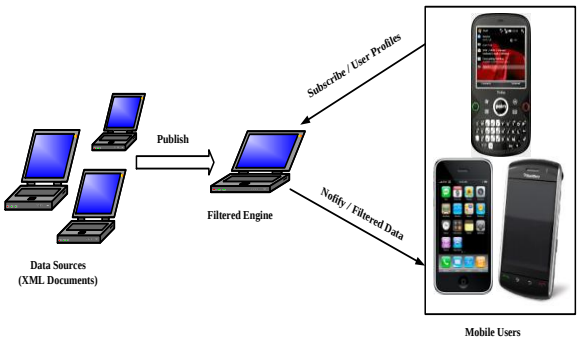


Figure 1. Overall architecture of the system

Data Sources contains the XML documents that publish messages to the filtered engine component of the pub/sub system. The mobile users register subscriptions (user profiles) that are provided graphical user interfaces to define their profiles from their mobile phones. These profiles are converted into XML-QL queries. The queries can be change based; that is, they need to be activated when the related XML documents change. The queries are grouped and indexed such that each element in a query group corresponds to a state in the Finite State Machine (FSM). Filtered engine first creates query indices for user profiles and then parses the XML documents to obtain the query results. When the document matches, the matches are stored in a special content list, so that the whole document need not be sent. Then, the filtered engine notifies and sends filtered XML documents to the related mobile clients. Extracting parts of an XML document can save bandwidth in a mobile environment.

Profiles defined through a graphical user interface are transformed into XML documents which contain XML-QL queries to provide user friendliness and expressive power as shown in Fig. 2.

```
<Profile>
   <XML-QL>
      WHERE <course>
                  <major>
                     <name>ICT</>
                     <program>First Year</>
                     <syllabus>$n</>
               </> </> IN “course.xml”
      CONSTRUCT<result><syllabus>$n</></>
   </XML-QL>
   <PushTo>
       <address>…</address>
   </PushTo>
</Profile>
```

Figure 2. Profile syntax represented in XML containing XML-QL query

A. Filtered Engine

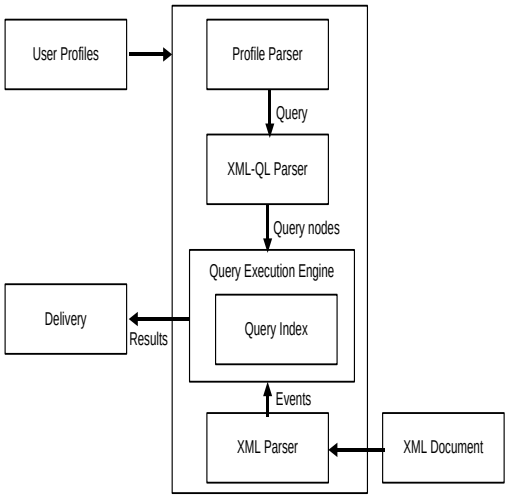


Figure 3. Filtered engine

The basic components of the filtered engine are 1) An event-based XML parser which is implemented using SAX API for XML documents; 2) A profile parser that has an XML-QL parser for user profiles and creates the Query Index; 3) A Query Execution Engine which contains the Query Index which is associated with Finite State Machines to query the XML documents; 4) Delivery Component which pushes the results to the related mobile clients. When a document arrives at the Filtered Engine, it is run through an XML Parser that then drives the process of query execution through the query index.

IV. Operation of the System

The system operates as follows: subscriber informs Filtered Engine when a new profile is created or updated; the profiles are stored in an XML file that contains XML-QL queries, execution conditions (change-base), and addresses to dispatch the results (see Fig. 2). The Profile Parser component of the Filtered Engine parses the profiles; XML-QL queries in the profile are parsed by an XML-QL parser. While parsing the queries, the XML-QL parser creates FSM representation of each query, if the query does not match to any existing query group. Otherwise, the FSM of the corresponding query group is used for the input query. FSM representation contains state nodes for each element name in the queries which are stored in the Query Index.

When a new document arrives, the system alerts the Filtered Engine so that the related XML document is parsed. The event based XML parser sends the events encountered to the Query Execution Engine (see Fig. 3). The handlers in the Query Execution Engine respond to these events. The handlers move the FSMs to their next states when current states succeed certain checks like evaluating the attributes, level checking or character data matching. In the mean time the data in the document that matches the variables are kept in content lists so that when the FSM reaches its final state, all the necessary partial data to produce the results are formatted and pushed to the related mobile clients.

A. Creating Query Index

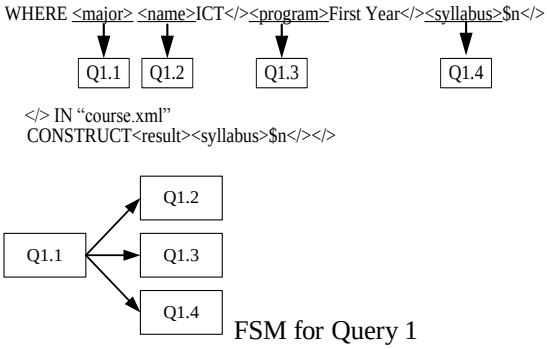
Consider an example XML document and its DTD given in Fig. 4.

```
<!-- DTD for Course -->
<!ELEMENT root (course*)>
<!ELEMENT course (degree, major*)>
<!ELEMENT degree (#PCDATA)>
<! ELEMENT major (name, program, semester,
syllabus*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT program (#PCDATA)>
<!ELEMENT semester (#PCDATA)>
<!ELEMENT syllabus (sub-code, sub-title, instructor)>
<!ELEMENT sub-code (#PCDATA)>
<!ELEMENT sub-title (#PCDATA)>
<!ELEMENT instructor (#PCDATA)>
<root>
<course>
  <degree>Bachelor</degree>
 <major><name>ICT</name>
      <program>First Year</program>
      <semester>First Semester</semester>
      <syllabus>
         <sub-code>ISCE-1001</sub-code>
         <sub-title>Web Programming</sub-title>
         <instructor>Dr. Thiri</instructor>
     </syllabus>
 </major>
</course>…</root>
```

Figure 4. An example XML document and its DTD (course.xml)

The example queries and their FSM representations are shown in Fig. 5. Note that there is a node in the FSM representation corresponding to each element in the query and the FSM representation's tree structure follows from XML-QL query structure.

Query 1: Retrieve all syllabuses for first year program in ICT major.



FSM for Query 1

Query 2: Find the name of instructor for subject code EM-101.

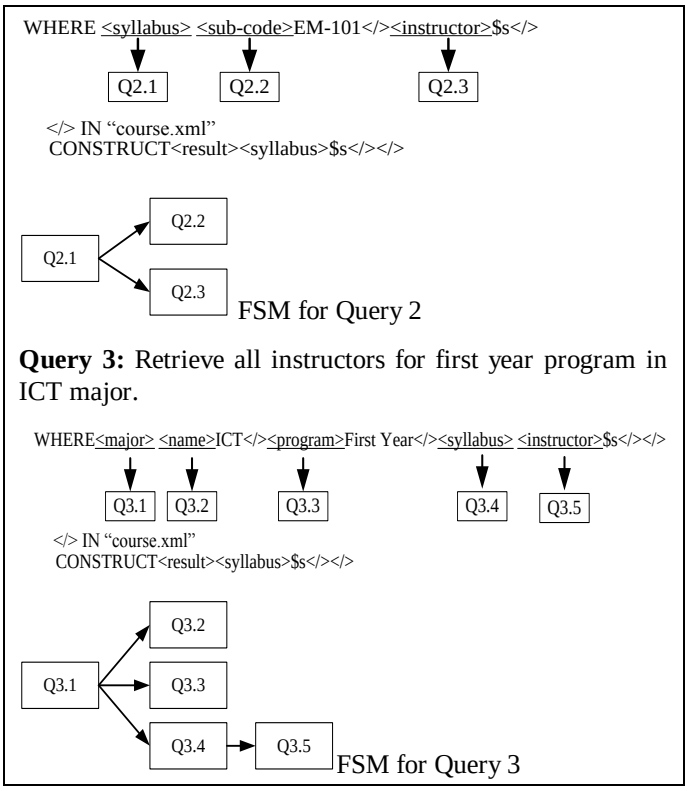


Figure 5. Example queries and its FSM representation

The state changes of a FSM are handled through the two lists associated with each node in the Query Index (See Fig. 6): The current nodes of each query are placed on the Candidate List (CL) of the index entry for its corresponding element name. All of the query nodes representing future states are stored in the Wait Lists (WL) of their corresponding element name. Copying a query node from WL to the CL represents a state transition in the FSM. Notice that the node copied to the CL also remains in the WL so that it can be reused by the FSM in future executions of the query since the same element name may reappear in another level in the XML document.

This system is intended to develop to handle very large number of queries and it is quite probable that there will be queries that have the same tree structure and the same element names, that is, the same FSM representation but different constant values. In this case a single FSM can handle these queries and this greatly enhances the performance of the system. When the query index is initialized, the first node of each query tree is placed on the CL of the index entry for its respective element name. The remaining elements in the query tree are placed in respective WLs. Query nodes in the CL indicate that the state of the query might change when the XML parser processes the respective elements of these nodes. When the XML parser catches a start element tag and if a node in the CL of the element in the Query Index satisfies level check and attribute check, and then the nodes of the immediate child elements of this node in the Query Index are copied from WL to CL. 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. The attribute check applies any simple expressions that reference the attributes of the element.

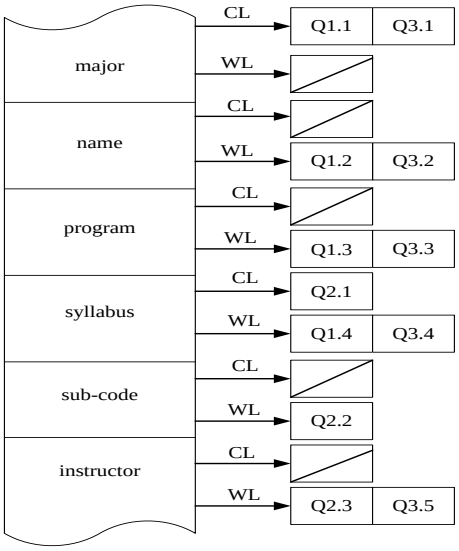


Figure 6. Initial states of the query index for example queries

B. *Operation of the Finite State Machine*

When a new XML document activates the XML SAX parser, it starts generating events. The following event handlers handle these events:

Start Element Handler checks whether the query element matches the element in the document. For this purpose it performs a level and an attribute check. If these are satisfied, depending on the type of the query node it either enables data comparison or starts variable content generation. As the next step, the nodes in the WL that are the immediate successors of this node are moved to CL at this stage. Even in a single document, the FSM may be executed more than once if the same element names reappear in the document. Therefore there is a need to reinitialize the FSM. Furthermore XML documents can be nested, that is, the same element may appear at different levels. Therefore it may be necessary to generate a FSM to handle this recursion. This is achieved by copying this new node to CL in the query index.

End Element Handler evaluates the state of a node by considering the states of its successor nodes and when the root node is reached it generates the output. End element handler also deletes the nodes from CL which are inserted in the start element handler of the node. This provides "backtracking" in the FSM.

Element Data Handler is implemented for data comparison in the query. If the expression is true, the state of the node is set to true and this value is used by the End Element Handler of the current element node.

End Document Handler signals the end of result generation and passes the results to the Delivery Component.

TABLE I. SAX API EXAMPLE

An XML Document	SAX API Events
<?xml version="1.0"> <course> <major> <name> ICT </name> </major> </course>	start document start element: course start element: major start element: name characters: ICT end element: name end element: major end element: course end document

C. Generating Customized Results

Results are generated when the end element of the root node of the query is encountered. Content lists of the variable nodes are traversed to fetch content groups. These content groups are further processed to generate results. This process is repeated until the end of the document is reached. The results need to be formatted as defined in the CONSTRUCT clause. The results of the queries that will be sent to related mobile phones.

V. Performance Evaluation

We conducted two sets of experiments to demonstrate the performance of the architecture for different document sizes and query workloads. The results are presented for a maximum of 250,00 queries (due to memory limitations), we expect that the performance of the system will still be acceptable for mobile environments for millions of queries since the results of the experiments show that the system is highly scalable. The execution time increases for all the algorithms because the queries become larger.

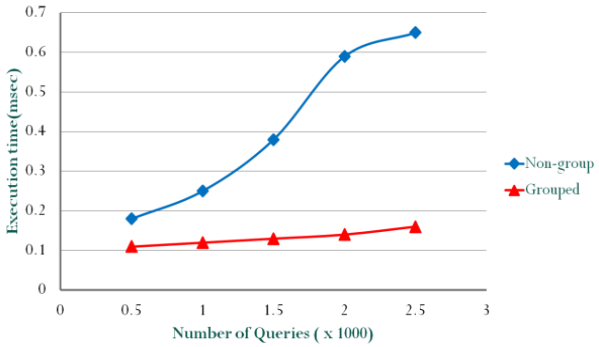


Figure 7. Comparing the performance by varying the number of queries

The graph shown in Fig. 7 contains the results for different query groups, that is, the queries have the same FSM representation but different constants, for the document course.xml (10KB).

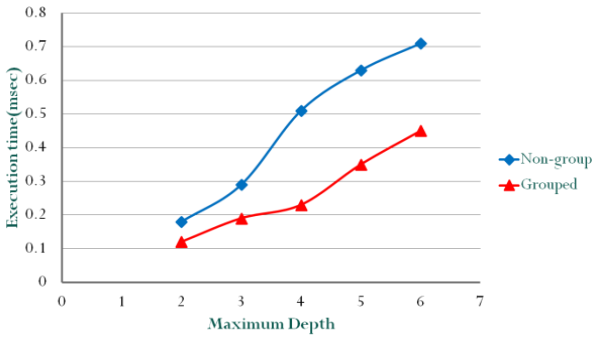


Figure 8. Comparing the performance by varying depth

The depth of the XML documents and queries in user profiles varies according to application characteristics. In this experiment, we evaluated the performance of the algorithms as the maximum depth is varied. Fig. 8 shows the filter time by varying depth of queries and documents. Our experimental result shows that the proposed algorithm outperforms the previous XFilter algorithm in time aspect.

VI. Conclusion

This paper attempts to develop an efficient and scalable SDI system for mobile clients based upon their profiles. In this paper, we described the architecture for XML-based pub/sub system built on top of a mobile ad hoc network. We propose an efficient indexing mechanism and a matching algorithm that can handle very large number of queries and achieve a good scalable performance of the system. The modified FSM is used as a filtering engine that scans incoming XML documents and discovers matching queries. A querying power is necessary for expressing highly personalized user profiles and for the system to be of use to millions of mobile users, it has to be scalable. To achieve high scalability in this architecture, we index the user profiles rather than the documents because of the excessively large number of profiles expected in the system.

Acknowledgment

The authors wish to acknowledge the Institution namely University of Technology (Yatanarpon Cyber City) for providing us with a good environment and facilities like internet, books, computers and all that as our source to complete this research. Our heart-felt thanks to our family, friends and colleagues who have helped us for the completion of this work.

REFERENCES

[1] M. Altinel and M. Franklin, "Efficient filtering of XML documents for selective dissemination of information," Proc of the Int'l Conf on VLDB, pp. 53-64, Sept 2000. *(references)*

[2] Y. Diao, M. Altinel, M. Franklin, H. Zhang and P.M. Fischer, "Path sharing and predicate evaluation for high-performance XML filtering," ACM Trans. Database Syst., 28(4), Dec 2003, pp. 467–516. *(references)*

[3] Extensible Markup Language, http://www.w3.org/XML/.

[4] I. Miliaraki, Distributed Filtering and Dissemination of XML Data in Peer-to-Peer Systems, PhD Thesis, Department of Informatics and

Telecommunications, National and Kapodistrian University of Athens, July 2011.

[5] I. Miliaraki and M. Koubarakis, "FoXtrot: distributed structural and value XML filtering", ACM Transactions on the Web, Vol. 6, No. 3, Article 12, Publication date: September 2012. *(references)*

[6] J. Chen, D. DeWitt, F. Tian and Y. Wang, "NiagaraCQ: a scalable continuous query system for internet databases", ACM SIGMOD, Texas, USA, June 2000, pp.379-390. *(references)*

[7] L. Dai, C. Lung and S. Majumdar, "A XML message filtering and matching approach in publish/subscribe systems", publication in the IEEE Globecom 2010 proceedings. *(references)*

[8] L. Dai, XML Message Filtering and Matching in Publish/Subscribe Systems, Master Thesis, School of Computer Science, Carleton University, Ottawa, Ontario, Canada, Sept. 2009.

[9] P. Antonellis and C. Makris, "XFIS: an XML filtering system based on string representation and matching", Int. J. of Web Engineering and Technology, Vol. 4, Nr 1, 2008. *(references)*

[10] P.T. Eugster, P.A. Felber, R. Guerraoui, and A.M. Kermarrec, "The many faces of publish/subscribe", ACM Computing Surveys 35, pages 114-131, 2003.

[11] Panu Silvasti, Algorithms for XML Filtering, Department of Computer Science and Engineering, Aalto University publication series Doctoral Dissertations 85/2011. *(references)*

[12] http://www.w3.org/TR/1998/NOTE-xml-ql-19980819.

Yi Yi Myint is a PhD candidate from University of Technology (Yatanarpon Cyber City) from Pyin Oo Lwin, Mandalay Division, Myanmar. She got master degree in computer science from Computer University (Mandalay). The field of her thesis is performance analysis of filtering algorithms and indexing methods for matching XML documents and user profiles.