Efficient Filtering Algorithm for Scalable XML-Based Publish/Subscribe Mobile System

Yi Yi Myint, and Hninn Aye Thant

***Abstract*—**Extensible Markup Language (XML) filtering systems comprise an essential component of recent information-seeking applications. The inherent limitations of mobile devices require information to be delivered to mobile clients to be highly personalized consistent with their profiles. The publish/subscribe (pub/sub) model has gained acceptance as a solution for enabling Selective Dissemination of Information (SDI) to mobile clients. In this paper, we propose an approach that integrates pub/sub system and XML filtering to deliver notifications with personalized information from XML data sources to mobile clients. The proposed architecture describes an efficient indexing mechanism for filtering and matching of XML documents against a large number of user profiles expressed with the XML-QL language. Our proposed algorithm is modified the Finite State Machine (FSM) approach based on XFilter algorithm to be used in mobile environment achieving good scalability.

***Keywords*—** XML, User Profiles, SDI, Mobile Clients

I. Introduction

The development of the Internet and networking technologies made it possible to access increasing volumes of data in a convenient way. As a consequence of these advances, information dissemination applications are gaining popularity in distributing data to the end users. The adoption of XML [3] as a de facto standard for information exchange entails a widely accepted profile representation and the advance of query languages for XML data facilitates the development of more efficient filtering mechanisms. The number of applications using XML representation is growing rapidly, thus the development of XML filtering is becoming an essential need of different application areas such as pub/sub system, web services and peer-to-peer networks.

The pub/sub communication paradigm [4] can assist to provide scalable and efficient push-based delivery of messages between the various parties in a system. In a pub/sub system, subscribers state their interests (profiles) in an event or a pattern of events, and are asynchronously notified when publishers generate them. The key idea of an XML filtering system is to discover all the user profiles that match with a particular XML document. Data matching can be performed either at the data sources or at some centralized brokers.

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. Then the user profiles need to be defined through an XML query language. XML-QL [9] 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. 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, the proposed filtering algorithm 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 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.

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

Yi Yi Myint is with Faculty of Information and Communication Technology, University of Technology (Yatanarpon Cyber City) , Pyin Oo Lwin, Mandalay Region, Myanmar (e-mail: yiyimyint.utycc@gmail.com).

Hninn Aye Thant is with Faculty of Information and Communication Technology, University of Technology (Yatanarpon Cyber City) , Pyin Oo Lwin, Mandalay Region, Myanmar (e-mail: hninayethant@gmail.com).

in the number of active states increasing exponentially.

BFilter [7] proposes the filtering and matching of XML message by leveraging branch points in both the XML document and user query. It evaluates user queries that exploit rearward matching branch points to hold up more matching processes until branch points match in the XML document and user query. In this fashion, filtering of XML message can be performed more efficiently since the probability of mismatching is reduced. Several experiments [8] have been conducted and the outcomes show that BFilter has better performance than YFilter for complex queries.

FoXtrot [5] is a XML filtering system which integrates the strengths of automata and distributed hash tables for efficient filtering to create a fully distributed system. FoXtrot also describes different methods for evaluating value-based predicates. The performance evaluation demonstrates that it is able to index millions of queries achieving an excellent 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.

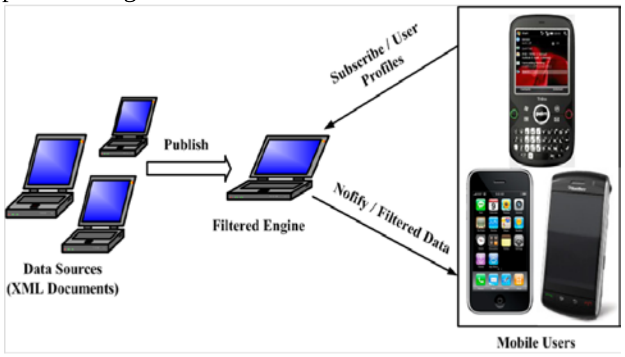


Fig. 1 Overall architecture of the system

Data Sources contains the XML documents that publish messages to the filtered engine. The mobile users register subscriptions in which the graphical user interfaces are provided for them to define user profiles. 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. The filtered engine notifies 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>$n</></>
                    </course> IN "course.xml"
      CONSTRUCT<result><syllabus>$n</syllabus></result>
   </XML-QL>
   <PushTo><address>…</address></PushTo>
</Profile>
```

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

A. *Filtered Engine*

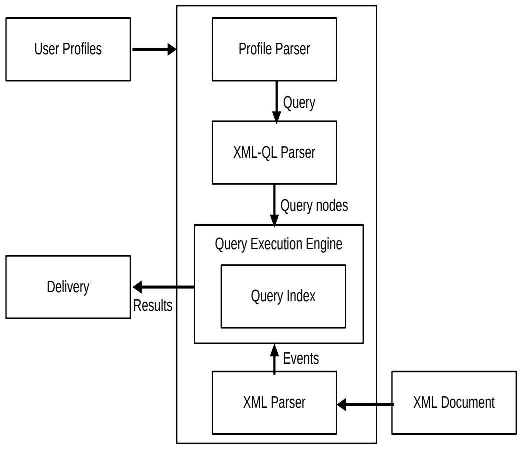


Fig. 3 Filtered engine

The basic components of the filtered engine are an event-based XML parser which is implemented using SAX API for XML documents, a profile parser that has an XML-QL parser for user profiles which creates the query index, a query execution engine which contains the query index which is associated with FSM to query the XML documents, and delivery component which pushes the results to the related mobile clients.

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 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 and move the FSMs to their next states when current states succeed 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 necessary partial data to produce the results are formatted and pushed to the related mobile clients.

A. *Creating Query Index*

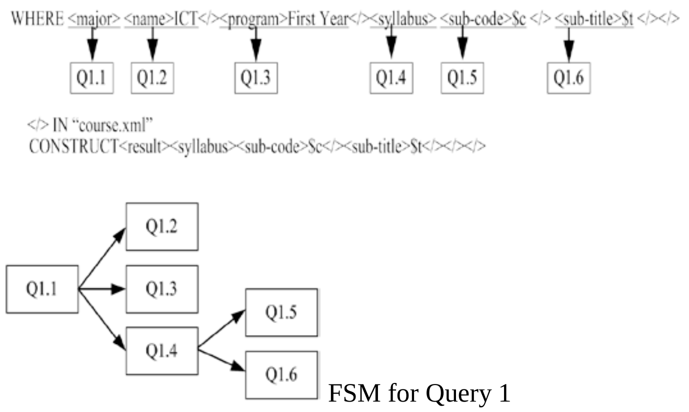
Consider an example XML document and its DTD given in Fig. 4.

```
<!-- 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>EM-101</sub-code>
              <sub-title>English</sub-title>
              <instructor>Dr. Thiri</instructor>
          </syllabus>
     </major>
</course>…</root>
```

Fig. 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.

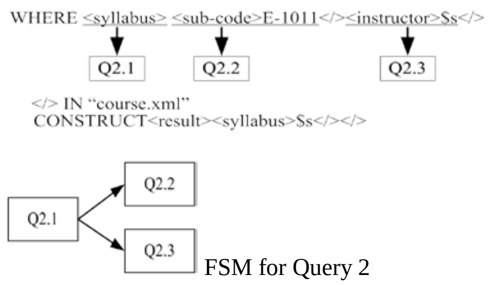
Query 1: Retrieve all syllabuses for first year program in ICT major.



```
WHERE <major> <name>ICT</><program>First Year</><syllabus> <sub-code>$c </> <sub-title>$t </></>
</> IN "course.xml"
CONSTRUCT<result><syllabus><sub-code>$c</><sub-title>$t</></></>
```

FSM for Query 1

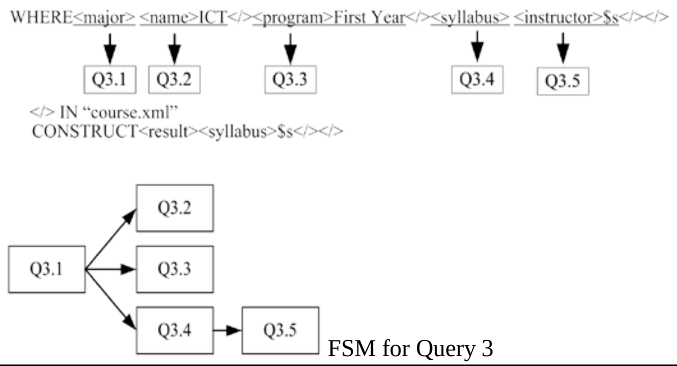
Query 2: Find the instructor name of the subject code E-1011.



```
WHERE <syllabus> <sub-code>E-1011</><instructor>$s</>
</> IN "course.xml"
CONSTRUCT<result><syllabus>$s</></>
```

FSM for Query 2

Query 3: Retrieve all instructors for first year program in ICT major.

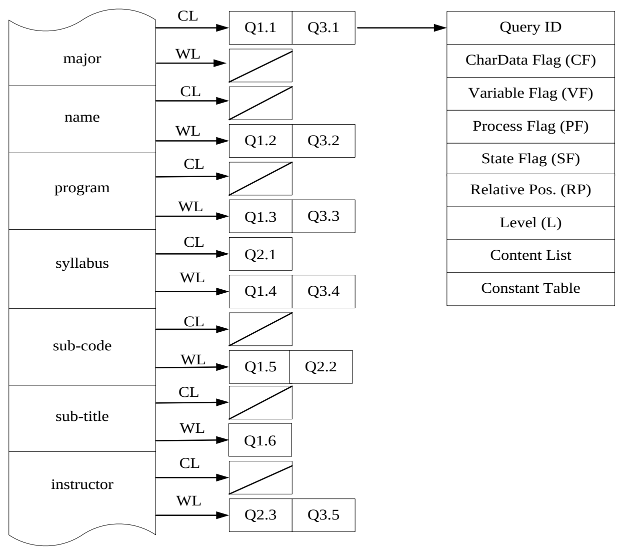


```
WHERE<major> <name>ICT</><program>First Year</><syllabus> <instructor>$s</></>
</> IN "course.xml"
CONSTRUCT<result><syllabus>$s</></>
```

FSM for Query 3

Fig.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.



(a)Query index (b) Node structure

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

The structure of the query index for the example queries as well as the structure of the query nodes is shown in Fig. 6.

Each node in each query has a unique identifier. Other elements of the node structure are as follows:

- Char Data Flag (CF) is on when the node in query has a character pattern to be matched.
- Variable Flag (VF) is on for nodes that have variables.
- Process Flag (PF) is used for dual purpose; it is set to true for nodes containing character pattern or containing variables.
- State flag (SF) is set to true when the element of this node is successfully processed level check, the attribute check or data comparison..
- Relative Position (RP) is an integer that describes the distance between this query node and the previous query node in a query tree.
- Level (L) is an integer that represents the level in the XML document.
- Content List stores intermediate results for a variable.
- Constant Table is used for queries that have same tree structure to hold the values of different constants.

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.

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:

TABLE I
SAX API EXAMPLE

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

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, 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.

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.

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.

C. *Proposed Filtering Algorithm*

Fig. 7 presents the filtering algorithm and depicts the procedure for filtering and matching XML documents.

```
Filtering Algorithm
 Input:
    QueryIndex qIndex
    Incoming element e
    List CurrentQueries Q
 Init:
   qIndex and Q is populated by user requests
  While e is not the end of document
     If e is in index then
        If node level= -1 || node level=element level then
            Node ok;
            If final node in query then
                Query match;
            Else if node level ≠ -1 then
                Update its level;
            End if;
        End if;
     End if;
  End while;
End Algorithm;
```

Fig. 7 Filtering algorithm

D. *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 clients.

V. PERFORMANCE EVALUATION

In the following experiments, we compare our incremental grouping approach with a non-grouping approach to show benefits from sharing computation and avoiding unnecessary query invocations. We conducted two sets of experiments to demonstrate the performance of the proposed architecture for different query workloads. The parameters in our experiments are:

1) N: The number of queries is an important measurement of the system scalability.
2) F: The number of fired queries in the grouping case.
3) S: The data size of XML document.

In our grouping approach, a user-defined query consists of grouped part and non-grouped part *Tg* and *Tng* represent the execution time of each part respectively. The execution time *T* for evaluating N queries is the sum of *Tg* and *Tng* of each of F fired queries,

$$T = Tg + \sum Tng \qquad (1)$$

The graph shown in Fig. 8 contains the results for different query groups for the document course.xml (12KB). The execution time of the non-grouping approach grows dramatically as N increases when setting F = N, i.e. all queries fired in both approaches.

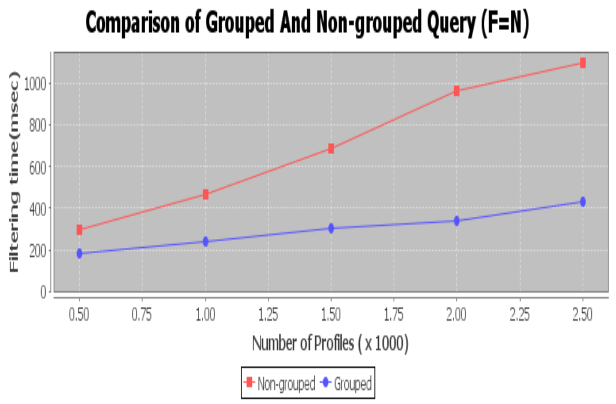


Fig. 8 Comparison of grouped and non-grouped query (F=N)

Fig. 9 shows the filter time by setting F = 100, i.e., 100 queries are invoked in the grouping approach.

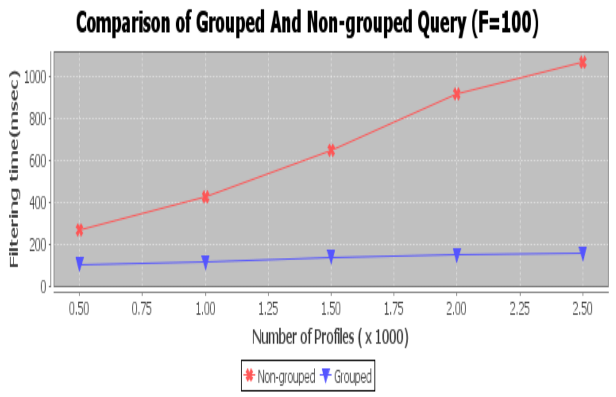


Fig. 9 Comparison of grouped and non-grouped query (F=100)

The filtering time of the grouping approach depends on number of fired queries F, not on the total number of installed queries N. The reason is that, although *Tg* increases as N grows, this shared computation is executed only once and is a very small portion of total filtering time. On the other hand, the filtering time for the non-grouping approach is proportional to N because all queries are scheduled for execution. Accordingly, 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.

VI. Conclusion

As the Web is growing continuously, a great amount of data becomes available to users, making it more difficult for them to discover interesting information by searching. In this paper, we propose an approach that integrates pub/sub system and XML filtering to deliver notifications with personalized information from XML resources to mobile clients. The proposed system describes an efficient indexing mechanism and a filtering algorithm based on a modified FSM approach that can quickly locate and evaluate relevant profiles. The proposed algorithm can handle very large number of queries and achieve a good scalable performance of the system. Our experimental result shows that the proposed algorithm outperforms the previous XFilter algorithm in time aspect.

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) http://dx.doi.org/10.1145/958942.958947

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

[4] 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. http://dx.doi.org/10.1145/857076.857078

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

[9] XML-QL: A Query Language for XML. [Online]. Available: http://www.w3.org/TR/1998/NOTE-xml-ql-19980819.

Yi Yi Myint is a PhD candidate from University of Technology (Yatanarpon Cyber City), Pyin Oo Lwin, Mandalay region, Myanmar. She got master degree in computer science from Computer University (Mandalay). The field of her thesis is performance analysis of filtering algorithm and indexing methods for matching XML documents and user profiles. Her research interested areas are mobile computing, ubiquitous computing and location-based services.