Synthesis Paper - Efficient Filtering of XML Documents for Selective Dissemination of Information

Eran Chinthaka (echintha@cs.indiana.edu)
Computer Science Department,
Indiana University,
Bloomington, IN 47405.

Abstract

With the rise of Internet and the wide adoption of distributed systems, the volume of information available ranging across various fields exploded. This also affected and increased the difficulty in surveying, querying and filtering information according to user profiles. The goal of SDI (Selective Dissemination of Information) systems is to provide information for the interested users via user profiles. Even though SDI systems existed in distributed systems even before internet, most of the complications became visible due to the adoption of Internet.

Lots of mechanisms were developed to implement these SDI systems and one widely used option is to express user interests using queries stored in user profiles. These queries are applied against all the incoming documents and the matched documents are delivered to the interested users.

This synthesis paper looks at three different approaches in implementing SDI systems to make this process efficient when the input documents are XML.

1 Introduction

The exploding amount of information available over the internet and increasing interest of users for this information fueled the development of various types of content dissemination systems. Some of the challenges of these systems included timely delivery of information, robustness, personalization, filtering against user profiles, etc., Initial SDI systems concentrated on the IR techniques from database technologies to match user profiles against new information. But database approaches and content dissemination approaches differ in a significant way. In a database system large numbers of data itesma re indexed and stored, and queries are applied individually. On the other hand, in SDU systems, large numbers of queries are stored and documents are individually matched against those queries. And also since the above mentioned approaches had less flexibility to express user interest, XPath and XQuery based systems began to gain acceptance across these systems.

The key challenge is to efficiently and quickly search the potentially huge set of user profiles to find those which the document is relevant. Initial systems developed using XPath have tried to apply each and every XPath to each and every document. But when the amount of information to be handled becomes large, these algorithms simply didn't scale. Researchers in SDI started searching for various ways to optimize the matching of these queries against a given document. This synthesis paper concentrates on three such prominent efforts.

Mehmet Altinel and Michael Franklin proposed one of the best XML filtering techniques, XFilter, in "Efficient Filtering of XML Documents for Selective Dissemination of Information" [1]. Chee-Yong Chan et al [2] came up with an improved version of XFilter, named XTrie, which outperformed XFilter both in terms of memory and speed.

A year later Michael Franklin and Yanlei Diao published a paper [3] which capitalized on the above efforts to provide more flexibility to the user using XQuery and shared path processing.

2 Overview on Papers

The SDI engine, XFilter, proposed by Altinel and Franklin provided one of the first implementations of content dissemination systems based on XML document and XPaths. They have concentrated on using XPaths over other querying techniques, because

1. They didn't require the full functionality provided by the other contemporary approaches, especially because SDI systems always tries to filter only one document at a given moment
2. XPath already had a specification from W3C, compared to other query languages.

XFilter first processes all the existing queries, which are found in user profiles, and decompose those queries into a set of nodes. These nodes represent the element nodes in the query and are considered as the states of a Finite State Machine (FSM). FSM approach is very important in an XPath based query processor especially because the order of the elements within user profiles should also be preserved.
Then it employs a query index on all the FSMs to process all queries simultaneously. The basic way to build this query index is to come up with a hash table which has the current element name as the key and next possible events as the value. When an XML document comes, it gets events from that document and those events drive the FSM to find the matches.
XFilter system had used two sophisticated methods, beyond the basic algorithm explained above to improve the efficiency of query indexing. Then the basic system was compared against these two improvements varying workload parameters like number of profiles, depth of elements of XML, level of element nodes in the XPath queries, etc.,

The XTrie paper is built on top of the XFilter approach and claimed 2-4 times improvement in speed over the XFilter system. XFilter relies on indexes created using the names of the XPath elements. But XTrie indexes on substrings, sequences of element names, and claim to provide much effective mechanism to index than XFilter.
The three key prominent features of XTrie can be summarized as follows.

1. XTrie can filter based on complex and multiple path XPath expressions
2. XTrie support both ordered and un-ordered matching of XML documents
3. Since XTrie uses substrings, instead of elements names to index, the authors claim that XTrie can reduce both the number of unnecessary index probes and avoid redundant matching.

Authors of this paper had also used an almost similar experimental setup to the one mentioned in XFilter. Chan et al had compared the approach proposed in this paper against two possible cases defined in XFilter paper, varying the same workload parameters mentioned in XFilter paper. The two approaches mentioned in XTrie paper, lazy XTrie and eager XTrie both out performs XFilter in every aspect of the observed

parameters. Even though there are no empirical evidence given in the paper, authors claim that XTrie is roughly about 33% memory efficient than XFilter. Since memory efficiency is a required property in a system which subjects to heavy concurrent loads and also, since XTrie is faster than XFilter, XTrie can be regarded as a more appropriate solution than XFilter for a SDI system.

The two systems discussed above send out the whole document to the user after a match of a profile against a given XML document. The third paper from Yanlei Diao and Michael Franklin talks about one more stage of a SDI system, in addition to the query processing system. The system proposed in that paper also concentrates on efficiently customizing the output message that will be sending to the users after successful matches.

This can be challenging than any of the above systems as this system should perform more tasks at least equal to the performance numbers of the above systems. The approach proposed in the paper is to allow the user not only to express their queries to select documents but also to customize the output using XQuery [6]. Since one XQuery might contain multiple queries and there can be several hundreds or thousands of such XQueries, the first two mentioned approaches might not perform well in this situation. So the authors have used YFilter [7], which is a high-performance shared path engine. YFilter uses Non-Deterministic Finite Automation to represent full set of navigation paths, and supports shared processing of the path expressions.
This paper builds on YFilter and has tried to exploit on shared processing of path expressions and to find an efficient way to perform post-processing of the results. Since the above two aspects are two competing points as aggressive path sharing requires more sophisticated post-processing, the authors have proposed a compromised solution, with the experimental results presented in the paper.

3 Taxonomy

The three papers mentioned in Section 2 are evaluated against following criteria to come up with taxonomy.

1. Scalability and robustness
2. Performance evaluation
3. Experimental setup
4. Assumptions
5. Organization and readability
6. Applications and future work

SDI systems, when integrated in to a distributed system, have to be robust and scalable with the increase of work load. Since these two aspects can be regarded as the key factors of a distributed SDI system, section 3.1 concentrates on evaluating the robustness and the scalability of the three systems proposed in these papers.
Section 3.2 looks at the experimental setup of these systems which are used to evaluate the performance. Since all three papers claim that the system proposed are performing well, it is worth to look at the base of experimental setups that are being used.
Section 3.3 discusses the results of the performance evaluations. These results are

interpreted in different perspective and this section tries to rationalize the claims that they have made by trying to interpret their conclusion on results.
Section 3.4 will focus on various assumption made in these systems. It will also discuss about the usability of the proposed system, in various SDI environments.
Section 3.5 talks about the overall organization of these papers and my views on readability of those, as a novel reader. Finally section 3.6 will look at the application and future works of the proposed system in a real world system.

3.1 Scalability

The authors of XFilter paper had tested their system with considerable amount of XML documents. But the paper had failed by not talking about the scalability and robustness of the system. Even though the number and variations of the input XML documents are large, there were no experiments done on the ability of this system to be used in a real time system. A usual SDI system will get hundreds or thousands of XML documents per second and it is very important to check the robustness of the system in such an environment. At the same time users might change their preferences, new users will come in and new components and modules will start sending events to the system. So the systems should be able to handle increased loads up to an acceptable level. It could have been better if the system was tested under multiple concurrent loads and this seems to be a problem with the XTrie paper as well. XTrie paper also used an extensive amount of XML documents, but the authors were not able to test the system under concurrent loads.

The authors of XML message brokering paper had done very good amount of work to test the scalability of the system. The researchers had setup a separate experiment to measure the effect of MQPT (Multi-Query Processing Time) on the variation of number of queries from 5000 to 40000. All the techniques that the paper discusses were tested with these settings.
The scalability results provided in the XML message brokering paper seems to be showing a clue about a problem in the scalability of the system. According to the figure 11, MQPT increases linearly with the number of queries, but this increase has a very high slope compared to the other tests given in this paper. This sharper increase can be due to the additional impact of recursive data on post-processing costs, as given in the paper. But this might be an alarming signal about the proposed system when it is to be used in a real system, where any kind of XML document can occur. So I think it is worthwhile to test this system against more complex and concurrent loads before really putting in to production.
None of the papers have estimated an upper bound of the loads that these systems can handle. Since the decision to employ a system in a production system also depends on their threshold values, it could have been better if the threshold values also were published.

3.2 Experimental Setup

Both XFilter and XTrie papers have used a standard set of DTDs from NITF (News Industry Text Format). The NITF DTD is mean to be used for press releases, wire services, broadcasters, newspapers and even for Web-based news organizations. This format is supported by various organizations as well. Since these systems are about

dissemination of information, I think the test document is a good format to be used. In addition to that both the systems have used same tools and methods to generate sample queries and XML documents. Even the parameters that are varied during the evaluations are the same for both experiments. So the comparisons found in XTrie paper can be regarded as a very good setup.
The XML broker paper also has used standard DTDs from the XQuery specification. But I think these two DTDs may not be a best test data for a content dissemination system. If the author had used more relevant DTDs for SDI system, the results of this experiment could have easily been mapped to a real time system.

3.3 Performance Evaluation

The authors of XFilter paper have tested their system for various workloads and optimization techniques. And the best technique, the list balance and pre-filtering technique, out of all they have tested, is also proven using empirical evidences and claimed in the conclusion.
But they haven't tested or done any performance evaluation with other systems to compare their systems against existing systems. They have claimed throughout the paper that the system is efficient, but not comparing it to an existing systems puts up a question about the amount of efficiency the system provides over the existing systems. Before this experiment, Tak W. Yan and Hector Garcia-Molina have proposed a similar content dissemination system [4], SIFT, which the authors of this paper could have used for performance comparisons. In addition to that, Marcos and Roberto et al [5], have also proposed a basic content dissemination system for publish/subscribe systems.
The XTrie paper, in contrast to the above paper had done a very good amount of work, by at least comparing their system against XFilter system. They have used the same experimental setup as in XFilter and have compared their results against it. According to the results given in the XTrie paper, on average, XTrie is 2-4 times faster than XFilter and about 33% efficient, in terms of memory.
But even though XTrie paper compares with the XFilter system, they haven't used the best case of XFilter for comparisons. As mentioned above also, the best case of the XFilter is when they use both list balance and pre-filtering. But XTrie authors haven't used it to compare their results which I think is a questionable.
The XML message broker paper, from the same second author of the XFilter paper, was unable to compare with an existing system. The performance numbers they have provided based on the shared path matching technique that they have proposed is promising. But the lack of data to compare it with existing systems might not give full credit to the efforts of this paper.

3.4 Assumptions

The three papers discussed so far have used various assumptions in the papers. This section will look at the viability of some of those assumptions.
All three papers are relying on pre-processed XPath and path expressions. This is important in a high work load environment as indexed queries will perform better and will not be a burden on the processing power of the system. Especially XFilter and XTrie systems compile the user queries and try to optimize the application of those queries to source data in run time.

But this assumption might not hold in a highly dynamic environment. For example, if these systems are used within a peer-to-peer system to filter out messages, there can be problems updating the compiled queries as and when peers go down or new peers join. One of the advantages of the proposed systems is that the approaches described do not require re-compiling existing set of queries when a new query or a profile comes in. But there can be a processing overhead on the system when the system is updated to a new query. At the same time if a user changes his preferences or removed itself from the system, then system must look at in every index and remove/update relevant entries. This can definitely be a performance problem, as the pre-compiled structure can no longer be used to apply against the incoming documents until it is updated to suit new conditions.
None of the papers have looked at this aspect of a SDI system. Even though this is highly probable in a peer-to-peer system, this situation can also happen, with a less probability in other systems as well. So evaluation of these systems under these conditions might give a more accurate picture of these systems.

Performance evaluation of the third paper clearly shows that all the best possible optimizations are not available and the whole system can be much slower when there is no DTD found for the exchanged messages. This can be a problem when proposed system is used in a Web services environment, which is the case when it comes to an event based publish/subscribe system used in LEAD (Linked Environment for Atmospheric Discovery) project. Even though there is a defined schema for SOAP messages, there is no single schema for the content of the body of the SOAP messages that are being exchanged through the SDI system. Even if the schema is defined in WSDL files of the operations that are invoked in this setting, there are large number static as well as dynamic WSDLs that can appear in this system. So some of the optimizations proposed in this system might not work in such an environment.

All these system are based on SAX to generate events for the source XML documents. The problem with SAX is that when it starts sending events for a given document, it cannot be stopped until the whole document is done with parsing. Even the above systems are done with processing all the indexed queries, those systems have to parse the whole document. There are few disadvantages of using SAX alone in a high-performance system like this.
First, if most of the queries can be evaluated by looking at the initial parts of the XML, then passing the whole document might be in-efficient. Again this can be a problem in a Web services environment. Most of the queries that user write will be applied on the "Header" element of the SOAP message. For example, a user might want to filter out messages that are destined to a particular endpoint. In that case employing some sort of "lazy" parsing might improve the efficiency of the system.
Second, if the XML that are exchanged are couple of MB or GB each, then consuming one XML file might take some time. If the user's query only checks the properties that are found in the initial parts of the XML, as discussed above, then parsing this big XML file using SAX will be a disaster for the system.

3.5 Organization and Readability

All three papers have started with a very good introduction of the background knowledge required to understand the concepts and have proceeded explaining the respective systems. XTrie paper seems to be bit complicated when it starts to explain on the underlying theory and optimizations behind the proposed system. But the authors of XFilter paper had explained all most the same theory in a more concise and easier way to understand to the reader. On the other hand XML message broker paper had done very good job on explaining the key concepts underneath the proposed system.

Every paper had related their work to the relevant works that were done on or before respective paper's efforts. But the XFilter seems to have not looked at the systems that are directly related to the problem that they are solving. As mentioned earlier also, there were some efforts [4] [5] on content dissemination by application of XPath expression, even though they are not as optimized as XFilter.

3.6 Applications and Future Work

This section will briefly look at how XFilter, XTrie and the XML message broker can perform in some of the practical content dissemination environment and possible improvements these systems might require to perform even more in the present environments.

Web services event brokers normally work on top of standard WS-Eventing protocol [8]. In a large scale system a SDI system is subscribed to all the event generators and Web services client are subscribed to this SDI system.

As described in the Assumption section also, none of these systems will be able to achieve the highest performance with the assumptions given in the respective papers. This is especially because user requirements can be highly dynamic and they might add, edit, and delete the queries more often. It is better if there was a study on these systems which measures the performance with the different kinds of changes in user queries.

At the same time, again as mentioned in the assumptions section, usage of SAX can be a problem to parse XML documents. The existing content dissemination implementation from Apache, the Apache Synapse project [9], has a simple implementation to apply XPath compared to the complex systems proposed in these papers. But when Web services environment becomes dynamic and overloaded it can be interesting to apply XFilter and XTrie to such an environment and how those systems can perform. One of the options that can be used to overcome this problem is to find the possibilities of plugging in StAX [10], Streaming API for XML as the XML parser to these systems. This might overcome the parsing of un-necessary parts of the XML improving the memory and performance of the system.

These concepts can also be used in an XML data base backed event triggering scheme as well. Especially the XML message broker proposed by Yanlei and Franklin will be useful as

1. It allows the customization of the output
2. The format of the XQueries are just like typical SQL queries
3. The indexing mechanism used in the system can be tightly integrated to the indexing scheme of the database.

It will be an interesting work to try XML message broker with such an XML database and come up with a performance evaluation.

Conclusion

XFilter paper and XTrie paper are two approaches for XPath processing of an SDI events. And the XML message brokering paper explains one of the pioneering efforts to have both filtering and output document customizations in a one single SDI system. All the systems are fast performing and have tried to exploit various aspects of XPaths, XQuery and XML documents. Had the papers explained deeper in to the respective areas through the aspects discussed in this paper, these experiments could have been more useful to be used in a real time system.

References

[1] M. Altinel, M. Franklin. Efficient filtering of XML documents for selective dissemination of information. In VLDB, Sep. 2000

[2] Chan, C.-Y., et al., Efficient Filtering of XML Documents with XPath Expressions. Proc. of 18th International Conference on Data Engineering (ICDE'02), 2002.

[3] Yanlei Diao, and Michael J. Franklin. Query Processing for High-Volume XML Message Brokering. In Proceedings of VLDB 2003, September 2003.

[4] Tak W. Yan and Hector Garcia-Molina The SIFT information dissemination system ACM Transactions on Database Systems (TODS) Volume 24 , Issue 4 Pages: 529 - 565 (December 1999)

[5] Marcos Aguilera, Robert E Storm, Daniel C Sturman, Mark Astley and Tushar D. Chandra Matching events in a content-based subscription system. Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing – 1999.

[6] S. Boag, D. Chamberlin, et al. XQuery 1.0: An XML query language. W3C Recommendation. http://www.w3.org/ TR/xquery, 2007

[7] Y. Diao, P. Fischer, et al. YFilter: Efficient and scalable filtering of XML documents. In *ICDE*, Feb. 2002.

[8] Don Box, Luis Felipe et al. Web Services Eventing 2006 http://www.w3.org/Submission/WS-Eventing

[9] Apache Synapse Project - A robust, lightweight implementation of a highly scalable and distributed service mediation framework based on Web services and XML specifications. http://ws.apache.org/synapse/

[10] Larry Cable, Thorick Chow et al. Java Specification Requests 173: Streaming API for XML http://jcp.org/en/jsr/detail?id=173