PARALLELIZED STRUCTURAL AND VALUE XML FILTERING ON MULTICORE PROCESSORS

Panagiotis Antonellis, Christos Makris and Georgios Pispirigos
Department of Computer Engineering and Informatics, University of Patras, Rio, Patras, Greece

Keywords: XML, Filtering, Parallel, Multi-core

Abstract: Information filtering systems constitute a critical component in modern information seeking applications. As the number of users grows and the information available becomes even bigger it is imperative to employ scalable and efficient representation and filtering techniques. Typically the use of XML representation entails the profile representation with the use of the XPath query language and the employment of efficient heuristic techniques for constraining the complexity of the filtering mechanism. However, most of the existing research work focuses on single-core systems, even though the multi-core processors are already widely used. In this paper we propose a parallel filtering algorithm based on the well known YFilter, which dynamically applies a work-load balancing approach to each thread to achieve the best parallelization. In addition, the proposed filtering algorithm extends YFilter to also support value-based predicates in the user profiles, thus enabling both structural and content-based XML filtering. Experimental results depict that the proposed system outperforms the previous parallel approaches to XML filtering problem.

1 INTRODUCTION

Information filtering systems (also known as publish/subscribe systems) (Aguilera et al., 1999) are systems that provide two main services: document selection (i.e., determining which documents match which users) and document delivery (i.e., routing matching documents from data sources to users). In order to implement efficiently these services, information filtering systems rely upon representations of user profiles, that are generated either explicitly by asking the users to state their interests, or implicitly by mechanisms that track the user behaviour and use it as a guide to construct his/her profile. Initial attempts to construct such profiles typically used “bag of words” representations and keyword similarity techniques (closely related to the well known vector space model representation in the Information Retrieval area) to represent user profiles and match them against new data items. These techniques, however, often suffer from limited ability to express user interests, being unable to fully capture the semantics of the user behaviour and user interests. As an attempt to face this lack of expressibility, there have appeared lately (Altinel and Franklin, 2000; Antonellis and Makris, 2008; Canadan et al., 2006; Diao et al., 2003; Kwon et al., 2005) a number of systems that use XML representations for both documents and user profiles and that employ various filtering techniques to match the XML representations of user documents with the provided profiles.

The basic mechanism used to describe user profiles in XML format is through the XPath query language (http://www.w3.org/). XPath is a query language for addressing parts of an XML document, while also providing basic facilities for manipulation of strings, numbers and booleans. XPath models an XML document as a tree of nodes. There are different types of nodes, including element nodes, attribute nodes and text nodes and XPath defines a way to compute a string-value for each type of node.

The process of filtering XML documents is the reverse of searching XML documents for specific structural and value information. An XML document filtering system stores user profiles along with additional information (e.g. personal information of the user, email address). A user profile can store either only structural criteria or both structural and value criteria. In the first case, the XML filtering is called structural while in the second case is called hybrid (structural and value-based). When an XML

Antonellis P., Makris C. and Pispirigos G..
PARALLELIZED STRUCTURAL AND VALUE XML FILTERING ON MULTICORE PROCESSORS.
DOI: 10.5220/0003896600050012
In *Proceedings of the 8th International Conference on Web Information Systems and Technologies* (WEBIST-2012), pages 5-12
ISBN: 978-989-8565-08-2

document arrives, the system filters it through the stored profiles to identify with which of them the document matches. After the filtering process has finished, the document can be sent to the corresponding users with matching profiles.

2 BACKGROUND

2.1 Related Work

In recent years, many approaches have been proposed for providing efficient filtering of XML data against large sets of user profiles. Depending on the way the user profiles and XML documents are represented and handled, the existing filtering systems can be categorized as follows:

Automata-based Systems. Systems in this category utilize Finite State Automata (FSA) to quickly match the incoming XML document with the stored user profiles. While parsing the XML document, each node element causes one or more transitions in the underlying FSA, based on the element's name or tag. In XFilter (Altinel and Franklin, 2000), user profiles are represented as queries using the XPath language and the filtering engine employs a sophisticated index structure and a modified Finite State Machine (FSM) approach to quickly locate and examine relevant profiles. A major drawback of XFilter is its lack of twig pattern support, as it handles only linear path expressions. Based on XFilter, a new system was proposed in (Diao et al., 2003) termed YFilter that combined all of the path queries into a single Nondeterministic Finite Automaton (NFA) and exploited commonality among user profiles by merging common prefixes of the user profile paths such that they were processed at most once. Unlike XFilter, YFilter handles twig patterns by decomposing them into separate linear paths and then performing post-processing over the intermediate matching results. The authors in (Zhang et al., 2010) propose a parallel implementation of YFilter for multi-core systems (shared-memory) by splitting the NFA into smaller parts, with each part assigned to a single thread. A distributed version of YFilter which also supports value-based predicates is presented in (Miliaraki and Koubarakis, 2010). In this approach the NFA is distributed along the nodes of a DHT network to speed-up the filtering process and various pruning techniques are applied based on the defined value predicates on the stored user profiles.

Sequence-based Systems. Systems in this category encode both the user profiles and the XML documents as string sequences and then transform the problem of XML filtering into that of subsequence matching between the document and profile sequences. FiST (Kwon et al., 2005) employs a novel holistic matching approach, that instead of splitting the twig patterns into separate linear paths, it transforms (through the use of the Prüfer sequence representation) the matching problem into a subsequence matching problem. In order to provide more efficient filtering, user profiles sequences are indexed using hash structures. XFIS (Antonellis and Makris, 2008) also employs a holistic matching approach which eliminates the need of extra post-processing of branch nodes by transforming the matching problem into a subsequence matching problem between the string sequence representation of user profiles and XML documents.

Stack-based Systems. The representative system of this category is AFilter (Canadan et al., 2006). AFilter utilizes a stack structure while filtering the XML document against user profiles. Its novel filtering mechanism exploits both prefix and suffix commonalities across filter statements, avoids unnecessarily eager result/state enumerations (such as NFA enumerations of active states) and decouples memory management task from result enumeration to ensure correct results even when the memory is tight. XPush (Gupta and Suciu, 2003) translates the collection of filter statements into a single deterministic pushdown automaton using stacks. The XPush machine uses a SAX parser that simulates a bottom up computation and hence doesn't require the main memory representation of the document. XSQ (Peng and Chawathe, 2005) utilizes a hierarchical arrangement of pushdown transducers augmented with buffers.

Although all of the previously described works have been used successfully for representing a set of user profiles and identifying XML documents that structurally match with the user profiles, little work (Kwon et al., 2008), (Miliaraki and Koubarakis, 2010) has been done to support value matching, that is evaluation of value-based predicates in the user profiles. This is a very usual problem in real world applications where the user profiles except for just defining some structural predicates, also introduce value-based predicates. A modern XML filtering system should be able to handle both types of predicates and also scale well in case of a large number of stored user profiles.

2.2 Paper Motivation and Contribution

Most of the research work in the area of XML filtering has been in the context of a single processing core. However, given the wide spread of multi-core processors, we believe that a parallel approach can provide significant benefits for a number of real world applications. In addition, most of the existing approaches concentrate only on the structural characteristics of user profiles, although in many real-world applications the value predicates may be more important.

Based on this motivation, we propose a parallel approach to the problem of structural and value-based XML filtering for shared-memory systems, based on the YFilter algorithm. The main contributions of the proposed parallel algorithm are:

- Parallel execution of the NFA constructed by the YFilter, by utilizing all the cores of the processor.
- Dynamic work load balancing based on the currently active states of the NFA.
- The support of value-predicates in user profiles, by dynamically pruning the NFA based on the most "popular" states.

In our knowledge, this is one of the few works in parallel XML filtering that deal with support of value-based predicates, mainly inspired by (Miliaraki and Koubarakis, 2010).

3 YFILTER OVERVIEW

The YFilter algorithm constructs a single NFA for a large number of queries and utilizes this NFA to filter a continuous stream of incoming XML documents (Diao et al., 2003).

In Figure 1 we present an example of such a nondeterministic finite automaton (NFA) constructed from four eight user profiles. The user profiles have been chosen appropriately to represent the different types of supported structural relationships. Each intermediate NFA state is represented with a circle, while each final NFA state (e.g. a state that leads to accepting a specific user profile) is represented with a double circle. The user profiles associated with each final state are shown with curly braces next to the state. Finally, each edge transition is triggered when a matching element (tag) name is encountered during the parsing of the incoming XML document.

As we can easily observe, YFilter greatly reduces the number of states by sharing the common prefix paths of the stored user profiles. YFilter uses an event-driven method along with a stack of active states. Each level of the stack represents possible states of the NFA for the XML part of the XML document that has currently already been seen. As shown in Figure 2, once it receives a start-of-element event, the filtering algorithm follows all matching transitions from all currently active states. When checking an available edge transitions, if the incoming element name matches the transition or the transition is marked by the * symbol, the corresponding state will be added to the new active state set. After all possible transitions have been checked, the new active state set is complete, and it is pushed on the stack as a new level. Whenever an accepting state is reached, it will output the user profiles list in this state. When an end-of-element event is received, the active states stack is popped one level.

It is vital to note that the actual operations required when a start-of-element event is received are checking the available transitions for each state in the top level of the active states stack. For example, when the start-of-element event for element <c> is received, the filtering algorithm checks the available transitions for the states: 2, 4, 5, 7 which result in the states 4, 6, 8, 9, 10, 11 to be activated and pushed in the top of the stack.

4 PARALLEL STRUCTURAL AND VALUE FILTERING

In this work we describe our new parallelized XM filtering algorithm, based on YFilter. The actual NFA execution is split into the different threads using a dynamic load balancing technique, which always ensures that each thread is assigned with the same work load. The proposed algorithm, in addition to structural filtering, also supports value filtering based on the value-predicates defined in the stored user profiles.

4.1 Parallelized NFA Execution

Our goal was to truly parallelize the YFilter algorithm in a balanced way in order for each thread to be assigned with a similar amount of workload during the filtering process. Existing approaches are based on statically splitting the constructed NFA into parts and assigning each specific part at each thread (Zhang et al., 2010). Similar approaches also exist for distributed NFA execution on top of DHT networks (Miliaraki and Koubarakis, 2010). However, this type of work splitting does not ensure

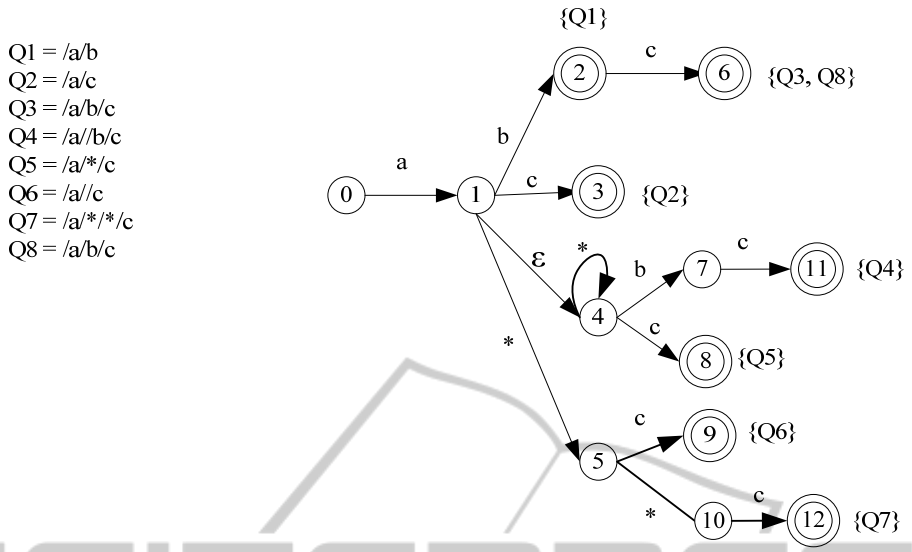

Figure 1: Example NFA constructed from a set of user profiles.

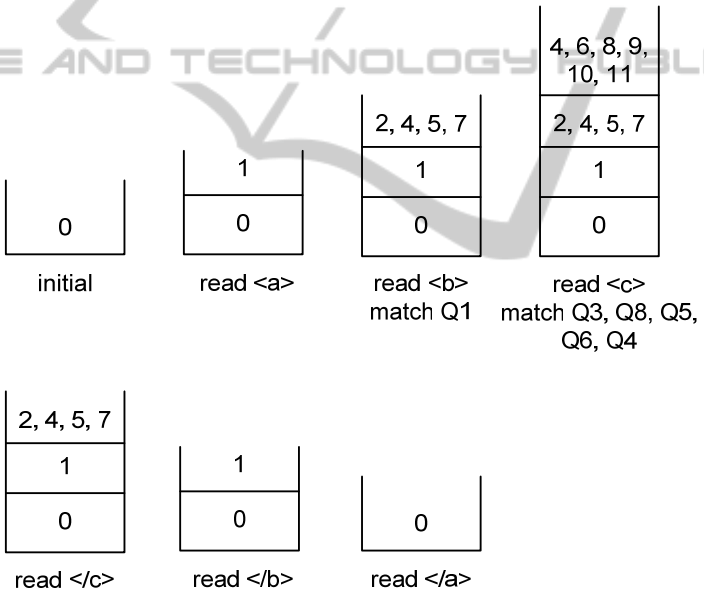

Figure 2: Active states during parsing of an incoming XML document.

that each thread will actually have the same workload, as the actual state transitions may occur only in a very small part of the whole NFA. In such a case some threads may remain idle, while others are working, thus the NFA execution is not truly parallelized. For example consider the NFA of Figure 1, split in four parts: {0, 1, 2}, {3, 4, 5}, {6, 7, 8} and {9, 10, 11, 12}, which each part statically assigned to a single thread. When the start-of-element event for element <c> is received, the thread #1 will check one state (state 1), the thread #2 will check two states (states 4, 5), the thread #3 will check only one state (state 7) and the thread #4 will remain inactive as none of the currently active states belong to its NFA part. Thus, the actual workload is not equally split to the four available threads.

Based on this notion, the proposed filtering algorithm achieves balanced work splitting, by

dynamically assigning tasks to each thread during NFA execution. As mentioned before, whenever a start-of-element event is received, the algorithm has to check the transitions of each state included in the top level of the active states stack. Although each active state does not have the same number of candidate transitions with the rest states, we make the assumption that the number of tasks is equal to the number of active states at that time. Based on this assumption, the proposed filtering algorithm creates a task for each active state and pushes them into a queue. Whenever a thread is idle, it is assigned with the next available task (e.g. active state) from the queue. The only drawback of this approach is that in cases that a state has a big number of candidate transitions, the corresponding thread may be late and thus the achieved parallelization will be not the best one.

For example, consider a system with three threads and the NFA of Figure 1, when the start-of-element event for element <c> is received. The currently active states are four (states 2, 4, 5, 7) and thus the thread #1 will be assigned with state 2, the thread #2 will be assigned with state 4 and the thread #3 will be assigned with state 5. Each thread will check the available transitions of its assigned state with the character <c> and in case of a match, it will activate the appropriate new state by pushing it on top of the stack. The first thread that will finish its job will be also assigned with the remaining state 7.

From the above example, it is clear that the proposed dynamic parallelization of the NFA execution achieves best results due to actual work balancing based on the currently active states, unlike the existing approaches which are based on statically assigning NFA subsets to each thread.

A slightly different approach can be utilized if the fan-out of the NFA (e.g. number of edges per state) is quite small: in such a case the actual cost of checking all the transitions of each state is quite small, so the overhead of creating a separate task for each active state may overcome the benefits of the actual parallelization. So, it is better to split the set of active states into a list of subsets, based on the number of threads, and assign a subset to each thread.

For example, consider again a system with two threads and the NFA of Figure 1, when the start-of-element event for element <c> is received. Instead of assigning the state 2 to thread #1, the state 4 to thread #2 and waiting for them to finish in order to assign the rest of the states, we can directly assign the states 2, 4 to thread #1 and the states 5,7 to thread #2. That way, we reduce the cost of task initialization for every separate active state, thus achieving a further improvement on the total filtering time.

This variation decreases the overhead introduced of task creation and assignment by creating the least number of tasks. However, if a specific subset of states includes much more transitions than the other subsets, the rest of the threads would have to remain idle for quite a long time, thus increasing the actual filtering time. Based on the above notions, the proposed filtering algorithm only uses this approach only if the number of transitions is about the same for every state during NFA construction, based on a predefined threshold of 15%, which was depicted after experimental testing.

4.2 Evaluation of Value-Based Predicates

In the previous section, we described how the structural matching is parallelized by assigning each thread with a subset of the active states to check. In this section, we concentrate on the evaluation of value-based predicates. Consider for example the user profile q:

paper[@year=2011]/author[text()="James"],

which selects the papers of author "James" during the year 2011. In order to filter an incoming XML document against the user profile q requires to check if the document's structure matches the profile's structure and also whether the value predicates of the user profile q are satisfied by the XML document.

A naïve approach is to integrate the value predicates directly on the constructed NFA, by adding extra transitions for the predicates, thus considering the value predicates as distinct nodes (Kwon et al., 2008). However, this approach would lead to a huge increase in the number of states and also destroy the sharing of path expressions for which the NFA was selected to begin with, as the value predicates usually form a larger set than the structural constraints of the user profiles. Other approaches, such as bottom-up and top-down (Miliaraki and Koubarakis, 2010), have been proposed to address this problem. The common idea behind those approaches is the selection of a small subset of the value predicates for pruning the NFA execution, based on some predefined selectivity criterion. However, in real world applications, the incoming XML documents have been usually generated by different sources and thus vary both in structure and content. In such cases, a selected value

predicate may be good for pruning the NFA execution during the filtering of a specific set of XML documents and bad for another set of XML documents. Thus, deciding on which value predicates to utilize during the NFA execution is not straightforward and has a strong impact on the efficiency of the filtering algorithm.

Based on this notion, our proposed filtering algorithm utilizes a novel step-by-step approach for supporting value-based predicates. This approach introduces the idea of "popular" NFA states, that is the NFA states that have been activated a lot during the filtering of the various incoming XML documents. More precisely, we keep a counter for each NFA state that counts the number of activations for that state and we select the top 10% states as the most "popular" states. For example, in Figure 3, the state 4 has been activated two times, while the state 3 has been activated zero times. The value of the threshold can change to balance the pruning of the NFA, but it is initialized to 10% which resulted in better results during the experiments.

The idea of utilizing the most "popular" states has the benefit that dynamically defines the set of NFA states that trigger value predicate checking (and thus may stop the NFA execution), only based on the set of previously filtered XML documents and not some user-defined selectivity criterion, like in (Miliaraki and Koubarakis, 2010). Thus there is no need for a-priori knowledge of the semantics of incoming XML documents in order to decide the those states. This approach is based on the idea that a state that has been activated a lot during the filtering of previous XML documents has a greater possibility to be activated during the filtering of subsequent XML documents, and thus an unsatisfied value predicate in that state will stop the NFA execution (prune this execution path).

During the NFA construction, at each state we also store a set of the corresponding value-based predicates along with the query id of each predicate. This set of predicates will be used during the filtering in order to decide whether the current execution path will continue or stop. Whenever an incoming XML document arrives, we parse it and create a list of candidate predicates based on the text data of nodes and attributes. This list of candidate predicates will be used during candidate checking during the filtering procedure.

Checking a set of predicates assigned to a state against the list of candidate predicates contained in an XML document may be a slow procedure, due to the big number of candidate predicates. Thus, instead of checking the predicates at each active state, the filtering algorithm applies the candidate checking only on the most "popular" states, as described before. During this check, we check if at least one of the state predicates is included in the list of document's candidate predicates. In such a case, the execution path will continue normally on this state. On the other hand, if none of the state predicates is part of the candidate predicates, then there is no need to continue this execution path as none of the corresponding user profiles are satisfied, thus the state is not activated.

The only drawback of this approach is that at the end of filtering process, all the matched user profiles must be checked against the incoming XML document based on their value-based predicates, as the filtering algorithm does not check the value predicates in all the states. However, usually the number of matched user profiles is a small portion of the total number of stored user profiles and thus the cost is very small compared to the cost of checking the value predicates at each NFA state.

5 EXPERIMENTS

We tested our filtering system against the most recent parallel approach to XML filtering (Zhang et al., 2010). In this approach the authors propose a method for statically splitting the NFA into subparts and assign each subpart to a separate thread. However, this approach does not support value-based predicates, so for the experiments we only used structural-only user profiles. Our filtering system was implemented in Java using the freeware Eclipse IDE. In order to obtain comparable and reliable results, we also implemented the other parallel algorithm in Java as well.

In our experiments we used three different datasets: DBLP dataset (http://kdl.cs.umass.edu/), Shakespeare's plays dataset (http://xml.coverpages.org/) and synthetic Treebank data generated by an XML generator provided by IBM (Diaz and Lovell). We also generated three user profile sets, one set for each dataset, using the XPath generator available in the YFilter package. The final set of user profiles consisted of the three different user profile sets, each set constructed from the corresponding XML dataset. We used that approach in order to emulate a real-world filtering system where the stored user profiles are usually different from each other and the same also stands for the incoming XML documents.

All the experiments were run on a quad-core hyper threading (thus 8 threads) Linux machine

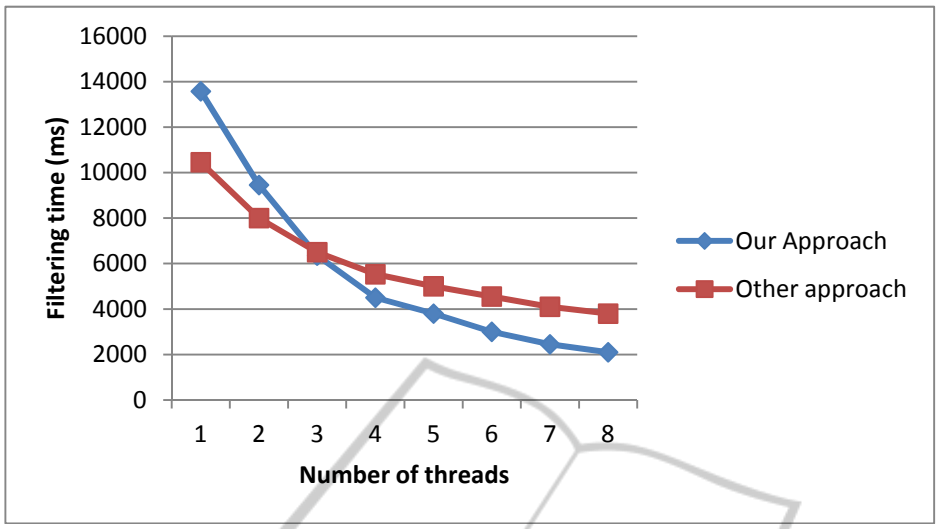

Figure 3: Filtering time as the number of threads increases.

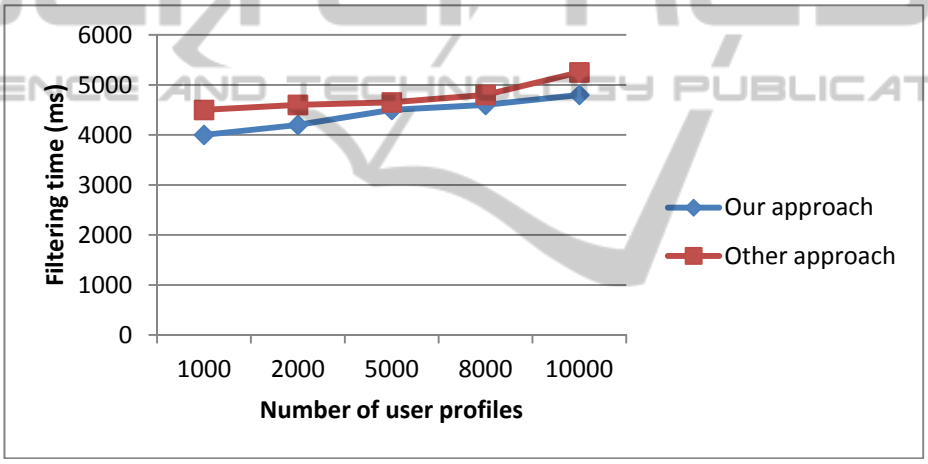

Figure 4: Filtering time as the number of user profiles increases.

running Kubuntu 11.04 with 8 Gb RAM. During the experiments, we measured the average filtering time of an XML document with size approximately 5500 nodes through 7000 stored user profiles, by varying the number of threads between 1 and 8, in order to calculate the speed-up gained by the proposed parallelization compared to the parallel algorithm presented in (Peng and Chawathe, 2005). In addition, we measured the average filtering time as the number of stored user profiles increases between 1000 and 10000, for 4 threads.

Figure 3 shows the results of the first experiment. As it can be easily observed both approaches achieve a speed-up of the total filtering process as the number of utilized threads increases. However, although our approach starts slower (for 1 thread), it turns out that it takes greater advantage of the increasing number of threads and finally achieves better filtering times after the 4 threads. In fact the achieved speed-up in filtering time is 7 (14000ms to 2000ms) for 8 threads, while the other algorithm actually achieves a speed-up of 2.5 (10000ms to 4000ms) for 8 threads. The results can be easily explained, as the overhead for creating a separate task for each active state can slow down the total filtering process if the number of threads is small (in the current experiment : 1- 3 threads), but as the number of threads increases, the proposed dynamic parallelization works efficiently and the total filtering time is greatly reduced. On the other hand, the approach proposed in (Peng and Chawathe, 2005), which is based on splitting the NFA into subsets and assigning each subset into a separate thread, cannot achieve the same speed-ups as the number of threads increases. This is due to the fact that it doesn't apply a balanced workload

splitting into the available threads, as each NFA subset execution may require different work, and thus some threads may remain idle for quite large amount of time

Figure 4 shows the results obtained from the second experiment. It is clear that the filtering time of both algorithms slightly increases as the number of stored user profiles increases. This is expected, as a greater number of user profiles results to a larger NFA and thus to a bigger number of active states during NFA execution. Thus, the actual workload increases and this is depicted in the total filtering time. However, the filtering time does not increase analogously to the total number of stored user profiles, which means that both approaches scale very well as the number of user profiles increases. Again, as the number of user profiles increases, our proposed parallel approach scales better than the algorithm proposed in (Peng and Chawathe, 2005), achieving an average of 15% better filtering time for 4 threads.

6 CONCLUSIONS

In this paper we have presented an innovative parallel XML filtering system that takes advantage of the multi-core processors that are widely used in modern computers, in order to speed up the XML filtering problem. The proposed system, which is based on the well-known YFilter algorithm, constructs a NFA from the stored user profiles and utilizes this NFA to filter a continuous stream of incoming XML documents. However, instead of executing the NFA using a single-thread approach, it splits the workload required at each step of the filtering process into the available threads, thus providing a big speed-up to the total filtering time required. The number of threads depends on the number of available cores and can vary, but the proposed filtering algorithm can work with any number of threads. In addition, the proposed filtering system extends the YFilter in order to efficiently support value-based predicates in the user profiles, enabling both structural and value-based filtering of the incoming XML documents. The value-based filtering is applied using a dynamic top-down approach, where the NFA execution is pruned only in the most popular states, which results to small overhead and big speed-up due to early pruning. The experimental results showed that the proposed system outperforms the previous parallel XML filtering algorithms by fully utilizing the available threads.

ACKNOWLEDGEMENTS

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

REFERENCES

Aguilera, M. K., Strom, R. E., Stunnan, D. C., Ashey, M. and Chandra, T. D. Matching Events in a Content-based Subscription System. *In Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC '99)*, 1999, 53-61.

Altinel, M. and Franklin, M.l J. Efficient Filtering of XML Documents for Selective Dissemination of Information. *In VLDB*, 2000, 53-64.

Antonellis, P. and Makris C. XFIS: an XML filtering system based on string representation and matching. *In International Journal on Web Engineering and Technology (IJWET)*, 2008, 4(1), 70-94

Canadan, K., Hsiung, W., Chen, S., Tatemura, J. and Agrrawal, D. AFilter: Adaptable XML Filtering with Prefix-Caching and Suffix-Clustering. *In VLDB*, 2006, 559-570.

Diao, Y., Altinel, M., Franklin, M.l J., Zhang, H. and Fischer, P. Path sharing and predicate evaluation for high-performance XML filtering. *In TODS*, 2003, 28(4), 467-516.

Gupta, A.K and Suciu, D. Stream processing of XPath queries with predicates. *In SIGMOD*, 2003, 419-430.

Kwon, J., Rao, P., Moon, B. and Lee, S. FiST: Scalable XML Document Filtering by Sequencing Twig Patterns. *In VLDB*, 2005, 217-228.

Kwon, J., Rao, P., Moon, B. and Lee, S. Value-based predicate filtering of XML documents. *In Data and Knowledge Engineering (KDE)*, 67 (1), 2008.

Miliaraki, I. and Koubarakis, M. Distributed structural and value XML filtering. *In DEBS*, 2010, 2-13.

Peng, F. and Chawathe, S. XSQ: A streaming XPath Queries. *In TODS*, 2005, 577-623.

Zhang, Y., Pan, Y. and Chiu, K. A Parallel XPath Engine Based on Concurrent NFA Execution. *In Proceedings of the IEEE 16th International Conference on Parallel and Distributed Systems (ICPADS 2010)*, 2010, 314-321.

http://www.w3.org/TR/xpath

http://kdl.cs.umass.edu/data/dblp/dblp-info.html

http://xml.coverpages.org/bosakShakespeare200.html

Diaz, A. L. and Lovell, D. XML Generator. http:// alphaworks.ibm.com/tech/xmlgenerator