RoXSum: Leveraging Data Aggregation and Batch Processing for XML Routing

CONFERENCE PAPER · JANUARY 2007

DOI: 10.1109/ICDE.2007.369037 · Source: DBLP

3 AUTHORS:

Zografoula Vagena
LogicBlox, Inc.
39 PUBLICATIONS **665** CITATIONS
SEE PROFILE

Mirella M Moro
Federal University of Minas Gerais
985 PUBLICATIONS **377** CITATIONS
SEE PROFILE

Vassilis Tsotras
University of California, Riverside
266 PUBLICATIONS **4,244** CITATIONS
SEE PROFILE

RoXSum: Leveraging Data Aggregation and Batch Processing for XML Routing

Zografoula Vagena
IBM Almaden Research Center
San Jose, CA 95120, USA
zovagena@us.ibm.com

Mirella M. Moro, Vassilis J. Tsotras*
University of California
Riverside, CA 92521, USA
{mirella,tsotras}@cs.ucr.edu

Abstract

Content-based routing is the primary form of communication within publish/subscribe systems. In those systems data transmission is performed by sophisticated overlay networks of content-based routers, which match data messages against registered subscriptions and forward them based on this matching. Despite their inherent complexities, such systems are expected to deliver information in a timely and scalable fashion. As a result, their successful deployment is a strenuous task. Relevant efforts have so far focused on the construction of the overlay network and the filtering of messages at each broker. However, the efficient transmission of messages has received less attention. In this work, we propose a solution that gracefully handles the transmission task, while providing performance benefits for the matching task as well. Along those lines, we design RoXSum, a message representation scheme that aggregates the routing information from multiple documents in a way that permits subscription matching directly on the aggregated content. Our performance study shows that RoXSum is a viable and effective technique, as it speeds up message routing for more than an order of magnitude.

1 Introduction

Content-based routing is a form of data delivery that differs from unicast, multicast and anycast communications, since the flow of messages is driven by their content rather than the IP address of their destination. This form of communication is widely employed by content-based data dissemination services. Such services (usually instantiated as publish/subscribe systems) enable *consumers* to describe the content of messages they are interested in (through user profiles), and *producers* to simply inject messages to the system without providing any addressing information.

*This research was partially supported by a UC Micro grant and Lotus Interworks; Mirella Moro was supported by Capes (Brazil).

Two important functions in those services are the *filtering of messages* (ie. matching consumers profiles to messages from producers), and the *routing of messages* (ie. defining the destination for each incoming message). Those tasks are usually left to the communication infrastructure, which typically consists of application-level routers (message brokers) organized in some overlay network structure.

With the adoption of XML as the standard for data exchange, XML-aware data dissemination services become necessary [2]. In those systems, the data to be routed are encoded as XML. User profiles can then be expressed using an XML query language. Recent research on XML-aware data dissemination services has focused on the filtering task, and various approaches discussing indexing of profiles [3, 2, 12, 8], batch processing of the incoming messages [6] and improving the filtering process [9] have appeared. For the routing task, several ideas have explored the construction of the routing tables [2, 14], and in situ transformation of the original data, in order to meet user requirements and improve transportation efficiency [2].

Nevertheless, the efficient transmission of messages has received less attention. Conventional data compression techniques have been proposed to achieve this task. Each message is compressed before leaving a broker and decompressed on arrival at another broker. However, as pinpointed in [14], the compression and decompression costs can significantly hamper the successful operation of such systems, and add up to the already large message filtering costs.

In this paper, we propose a new message representation scheme, coupled with novel filtering algorithms that combine the advantages of content aggregation and batch processing. We improve the overall effectiveness of the whole message routing infrastructure by designing a solution that decreases communication costs, while boosting the performance of the message filtering process. Our main contributions are summarized as follows.

- we design a new structure, the *RoXSum* (Routing XML Summary), which aggregates the content of multiple messages.
- we demonstrate that the message filtering process can

be performed on *RoXSum* itself, instead of the original messages, thus enjoying the merits of batch processing while decreasing decompression overheads.
- we devise incremental composition and decomposition algorithms for *RoXSum* to be performed at each broker.
- we demonstrate the efficiency of our solutions with a thorough empirical evaluation that unveils the characteristics of the proposed methods on a prototype message routing system that we have implemented.

We proceed with related work in section 2 and the *RoXSum* structure and its challenges in section 3. Section 4 describes the processing algorithms. Section 5 presents an experimental study that evaluates the performance of our techniques on a prototype message routing system and section 6 concludes the paper.

2 Related Work

A large number of proposals related to the efficient deployment of publish/subscribe systems have already appeared. An early system [18] provided support for matching keyword search queries over large collections of documents. Most systems employ the *Event-Condition-Action* paradigm to perform profile matching and selective dissemination of information. Events are usually described either as conjunctions of *(attribute, value)* pairs (eg. [4]), or in XML [16, 2, 12], and profiles are expressed as selection predicates over the content of events. For messages and user profiles represented in XML, automata-based profile algorithms are among the most popular message matching solutions [10, 3, 9]. Nevertheless, several alternative matching techniques, such as relational joins [16], bloom filters [8] and subsequence matching [12] have also been proposed. The goal of these works is scalability with respect to the number of user profiles, which is achieved by employing multi-query processing methods. The work in [6] targets scalability on the number of messages and designs matching techniques which handle message batches. Our work *combines* the advantages of both optimizations by employing multi-query processing on batches of messages.

While early publish/subscribe systems were centralized, scalability requirements have mandated the study of distributed architectures. SIFT [18] was the first system to provide solutions for distributed message filtering. Recently, works that deal with the construction of the overlay network structure [15, 5], the distribution of user profiles [2] and message routing policies (eg. [14]) have appeared. [15, 2, 5] use XML as the encoding format for messages, which is also the focus of this paper. Our solution is *complementary* to those works, as it targets the design of a compact but directly queryable message format. Moreover, it can be integrated with and benefit any of the previous architectures.

RoXSum is inspired by ideas from the summarization of XML data where structural constraints of the original data are preserved (e.g. [7, 13]). RoXSum uses the basic idea of identifying and representing the structural relationships of the original data. Nevertheless, it functions at the granularity of documents instead of document nodes. Hence, RoXSum is tailored for compactly representing information among *multiple* documents and is optimized for quickly identifying whole documents (rather than nodes).

3 The RoXSum Data Structure

A key component in our system is a new data structure called *RoXSum*. Its purposes are to aggregate the structural information contained in a set of XML documents, and to permit the message filtering process to be applied directly on itself, instead of the original documents. Its design is based on the observation that within an XML document (or among multiple XML documents), elements share structure and labels. This observation is the basis for all structural summary structures that have appeared in the literature and their effectiveness is well-established (eg. [7, 11]).

RoXSum is a data structure that aggregates a set of documents in such a way that their common parts are stored only once. It consists of two parts: a hierarchical structure called *RoXSum tree*, and a set of groups of documents identifiers called *RoXSum extents*. Each node in the *RoXSum tree* groups all *structurally equivalent* nodes from the document. The notion of *structural equivalence* is formalized with the concept of bisimilarity [13, 11].

Two document nodes correspond to the same node in the *RoXSum tree* if and only if they are bisimilar. Intuitively, two nodes on the same document are bisimilar if the sequences of labels of the incoming paths to the nodes are the same. It is not difficult to show that when the original document has a tree structure (as is the case in this paper) the derived structure is also a tree, thus the name *RoXSum tree*. To aggregate multiple documents within the same *RoXSum* structure, we assume the existence of a virtual root with edges to each one of the document roots.

Analogously to previous related results [13, 11], if a given path expression query is evaluated over the *RoXSum tree*, and the nodes that satisfy the query are identified in that tree, then the corresponding documents nodes, and only those nodes, are the ones that satisfy the same query. As a result, the *RoXSum tree* is both safe and precise; safe because it does not miss any result and precise because it does not produce any false positives.

For message filtering purposes, we need to identify the documents that satisfy a query. A naive solution is to associate the set of identifiers of those documents that contain the corresponding bisimilar nodes to the corresponding *RoXSum tree* node. So, after identifying the *RoXSum tree*

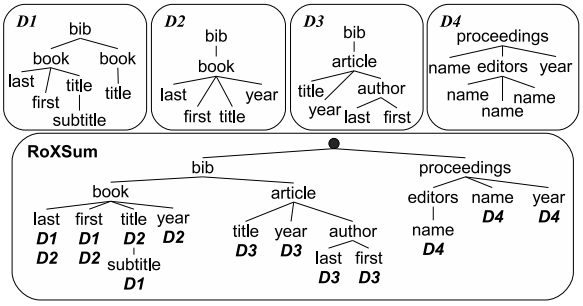

Figure 1. RoXSum Data Structure.

nodes that satisfy a query, we can obtain the documents that satisfy it as well. This solution produces the correct answer, but the total size of a *RoXSum* is proportional to "number of documents * the total number of nodes" (i.e. the sum of the number of nodes in each document). To decrease this space, we consider an important observation: each document stored on a *RoXSum* structure is a tree; so, identifying the leaf nodes of all root-to-leaf paths with a document *id* is enough to imply all internal path nodes of that document. Hence, we can associate a document id with *only* the *RoXSum tree* nodes that correspond to the *leaf* nodes of that document. The set of document identifiers that correspond to a *RoXSum tree* node is the *RoXSum extent* of that node.

An example *RoXSum* structure appears in figure 1. A collection containing documents with identifiers $D1$, $D2$, $D3$ and $D4$ is shown at the top of the figure, while the corresponding *RoXSum* structure is illustrated at the bottom. The document identifiers under each label in the *RoXSum* structure represent the extent of the respective node.

Having this structure, identifying documents (when evaluating a query or rebuilding the document) from the *RoXSum tree* is straight-forward. From each node on the *RoXSum tree* that satisfies a query, its extent and the extent of its descendants contains those, and only those, documents that satisfy the query as well. For example, consider the query */bib/book/title* on the documents of figure 1. There is only one path in the *RoXSum* that satisfies this query. All documents within the extent of the *RoXSum tree* node *title* (under elements *bib* and *book*), and within its descendant node *subtitle* satisfy the query, i.e. documents *D1* and *D2*. In the next section, we explain how *RoXSum* is used within our message routing infrastructure.

4 Data Dissemination Process

We assume the existence of a network of K content based routers (or *brokers*) whose purpose is to route incoming messages to their corresponding consumers[1]. The main components of our infrastructure are: *producers* that create XML documents and inject them in the routing system through an *entrance broker*; *profiles* defined by users as path expressions over the messages; a *routing table* that is maintained at each broker for storing all user profiles (on that broker) with respective target broker information; and *messages* (exchanged within the system) that have a header with control information, the *RoXSum tree*, and the text information (for matching value conditions if needed).

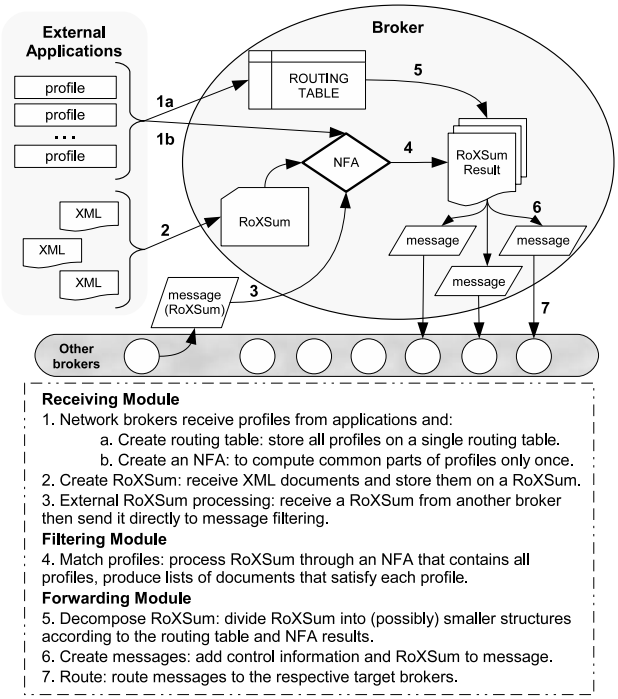

Figure 2. Data flow and main tasks within a broker.

At each point in time, a broker has to accomplish three main operations, namely: (i) receive incoming messages, (ii) match received messages against profiles, and (iii) forward messages appropriately. Each of these operations consists of smaller tasks performed by a distinct module within the broker. Figure 2 presents the flow of information and the operations within the processing modules of a broker. The responsibilities of each module are summarized as follows[2].

Receiving Module. This module receives profiles and groups them into the broker's routing table and the profile NFA, which is a Non-deterministic Finite Automaton that is used for multi-message filtering (task 1). This task is performed whenever a broker receives either new profiles to be added to its routing table or invalidation notices for some of the existing profiles. Furthermore, this module accepts streams of documents from client applications and is responsible for aggregating them, forming new *RoXSum* structures (task 2). *RoxSum* structures are created by processing the stream of incoming messages through a SAX-

[1] The actual construction and maintenance of the communication infrastructure is outside the scope of this paper and has been covered elsewhere (e.g. [15, 5]).

[2] Due to lack of space, complete descriptions and algorithms will appear in the extended version of this paper.

based parser. This parser reads the XML document in-order and either (i) adds new index nodes to the *RoXSum tree*, or (ii) adds new document ids to existing index nodes, following the bisimilarity constraint previously explained. Finally, this module forwards each incoming *RoXSum* to the filtering module (task 3).

Filtering Module. This module matches the profiles against *RoXSum* structures, one structure at a time (task 4). It employs a new automata-based, multi-query processing technique that operates on the *RoXSum* structure itself. Specifically, for each set of profiles, it is guaranteed that their common parts are processed only once by the NFA. Building the NFA resembles the approach in [3]. However there are two differences: (i) instead of operating at node granularity and on the original documents, our NFA is tailored to operate on document granularity on the RoXSum structure; and (ii) our implementation uses different structures to keep the results (optimized linked stacks and hash tables). All profiles are matched against all documents that comprise the structure, by performing single pass over a *RoXSum* structure. This process outputs a set of *(document identifier, destination broker)* pairs that are passed to the forwarding module.

Forwarding Module. This module sends each document to its destination broker. Given the results of the matching phase, this module has to accomplish three tasks: (i) gather the content of each document, (ii) aggregate the contents of documents that are sent to the same broker into a new *RoXSum* structure (task 5), and (iii) formulate the messages to be handed on the underlying network infrastructure (task 6). The first two tasks are performed on the *RoXSum* structure *itself* (through structure traversals, partitions and aggregations with other structures) thus avoiding the need to reconstruct the original documents. There are also two hash tables involved: *docTarget*, which maps each document id that satisfies any profile to the respective target brokers, and *targetRS*, which maps target ids to the root of the RoXSum that will be routed to them. Finally, task 7 is performed by the underlying network infrastructure.

For example, figure 3a depicts a given RoXSum structure and a routing table. The matching profile module returns that documents *D1* matches profiles */a/b/c* and */a//d/e*, and *D2* matches profile */a/b/c*. Then, *docTarget* contains two entries (one per matched document) as in figure 3b. At the end, *targetRS* is populated as in 3b. Each target receives a *RoXSum* with only those documents that match any of the respective profiles.

5 Experimental Evaluation

To empirically study the viability of our method, we built a simulator of a network of N brokers for manipulating RoXSum structures. This section presents a summary of

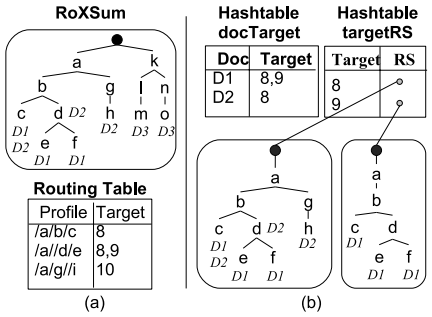

Figure 3. Decomposition of RoXSum structure.

the results of a series of experiments that were conducted to assess the behavior of the RoXSum structure.

We utilized both benchmark (XMark [17]) and custom generated datasets. The custom datasets represent highly heterogeneous collections which are common in distributed publish/subscribe systems. For the heterogeneous collections, we collected 22 different DTDs with variable structures and used them to generate the input documents, with the aid of the ToXGene XML document generator [1].

The experiments were conducted on an Intel Pentium IV, 2.6GHz machine, with 1GB of memory. All algorithms were implemented in Java using Sun JDK version 1.4.0.

Message Size. We compare the space requirements for routing a set of documents individually as opposed to aggregating them in a RoxSum structure. On sets with 100 up to 100,000 documents, the size of the *RoXSum tree* is usually less than 5% of the total size of the documents. Likewise, the size of the *RoXSum tree* with the documents content, is at least 10 times smaller than the regular documents.

Routing within a Broker These experiments evaluate the performance of the tasks that take place within a single broker. We measure the time between the moment a set of documents enters an entrance broker to the moment the same documents are forwarded to their target brokers. The main parameters that affect such performance are:

- Number of input documents: affects the amount of data to be processed and the size of the associated *RoXSum* structure.
- Number of target brokers: influences the number of output messages to be created. It also affects how many *RoXSum* structures will be created from the original structure. Additionally, the size of each structure is defined by a combination of the number of target brokers and the profile selectivity.
- Number of profiles: quantifies the system scalability regarding the number of profiles it can handle at the same time.
- Selectivity of the profiles: defines the amount of information from the input that needs to be routed, i.e. how many documents need to be sent to target brokers.

The results when varying those parameters are as follows.

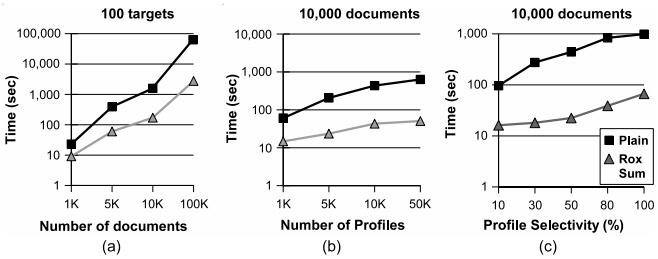

Figure 4. Performance when varying the number of input documents and target brokers - all in logarithmic scale.

Varying the Number of Documents for 100 Targets. We use $1,000$ profiles whose associated path expressions have from 3 up to 7 nodes each, and all incoming documents satisfy at least one of them. The profiles distribute the documents uniformly among the target brokers (we show the results for 100 target brokers but similar measures were obtained for 1 and 10 brokers). Figure 4a has the results for the number of documents varying from $1,000$ ($1K$) to $100,000$ ($100K$). The Y axis presents the time (in seconds), whereas the X axis presents the number of input documents evaluated. In all cases, except when evaluating few documents ($1K$), the *RoXSum* is at least *10* times faster than evaluating the documents sequentially. This difference practically reaches two orders of magnitude when evaluating $100K$ messages. This happens because the very small size of the *RoXSum* structure enables the profile evaluation and the output message processing to be performed very quickly.

Varying the Number of Profiles. Here, we keep the number of documents fixed in 10K, the selectivity of the set of profiles fixed to 50%, and the number of target brokers to 10. The message filtering results are uniformly routed to those 10 target brokers. The number of profiles varies from $1,000$ (1K) to $50,000$ (50K). Figure 4b illustrates the results. The Y axis presents time, whereas the X presents the number of profiles evaluated against the documents. We compare the results provided when matching the profiles to the *RoXSum* structure (which includes the time to build the *RoXSum*) versus to each document separately. In most cases, the evaluation on the *RoXSum* is around 10 times faster than the plain solution.

Varying Profile Selectivity. We considered 10 different target brokers and $1,000$ profiles on 10K documents. The selectivity is defined for the set of profiles (ie. a selectivity of 10% means that 1 in each 10 input documents satisfies *any* of the $1,000$ profiles). Figure 4c illustrates the results. The Y axis presents the time, whereas the X axis presents the selectivity. Once more, the results show that employing the *RoXSum* structure can result in an order of magnitude faster evaluation compared to the processing of each documents separately. Moreover, this experiment shows that the *RoXSum* performs gracefully when varying the number of both the input and the output documents.

6 Conclusion

In this paper we focused on the efficient routing of messages within a content-based routing system. We proposed *RoXSum*, a message representation scheme that aggregates the content information of different messages and enables very efficient profile matching. Moreover, its performance adapts gracefully to increasing number of both messages and profiles. *RoXSum* avoids document decompression at each broker by performing the subscription matching directly on the aggregated content while permitting batch processing for message filtering. Our experiments demonstrate that employing *RoXSum* within content-based message routing systems can significantly boost their performance and scalability.

References

[1] D. Barbosa et.al. Toxgene: A Template-based Data Generator for XML. In *WebDB*, 2002.

[2] Y. Diao, S. Rizvi, and M. J. Franklin. Towards an Internet-Scale XML Dissemination Service. In *VLDB*, 2004.

[3] Y. Diao et. al. Path Sharing and Predicate Evaluation for High-Performance XML Filtering. *ACM TODS*, 28(4), 2003.

[4] F. Fabret et. al. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe. In *SIGMOD*, 2001.

[5] W. Fenner et. al. XTreeNet: Scalable Overlay Networks for XML Content Dissemination and Querying. In *WCW*, 2005.

[6] P. M. Fischer and D. Kossmann. Batched Processing for Information Filters. In *ICDE*, 2005.

[7] R. Goldman and J. Widom. DataGuides: Enabling Formulation and Optimization in Semistructured Databases. In *VLDB*, 1997.

[8] X. Gong et. al. Bloom Filter-based XML Packets Filtering for Millions of Path Queries. In *ICDE*, 2005.

[9] B. He, Q. Luo, and B. Choi. Cache-Conscious Automata for XML Filtering. In *ICDE*, 2005.

[10] T. J. Green et. al. Processing XML Streams with Deterministic Automata. In *ICDT*, 2003.

[11] R. Kaushik et. al. Exploiting Local Similarity for Indexing Paths in Graph-Structured Data. In *ICDE*, 2002.

[12] J. Kwon et al. FiST: Scalable XML Document Filtering by Sequencing Twig Patterns. In *VLDB*, 2005.

[13] T. Milo and D. Suciu. Index Structures for Path Expressions. In *ICDT*, 1999.

[14] O. Papaemmanouil and U. Centintemel. SemCast: Semantic Multicast for Content-based Data Dissemination. In *ICDE*, 2005.

[15] A. C. Snoeren, K. Conley, and D. K. Gifford. Mesh-Based Content Routing using XML. In *SOSP*, 2001.

[16] F. Tian et. al. Implementing a Scalable XML Publish/Subscribe System Using Relational Database Systems. In *SIGMOD*, 2004.

[17] XMark. The XML benchmark project. In *http://www.xml-benchmark.org*.

[18] T. W. Yan and H. Garcia-Molina. The SIFT Information Dissemination System. *ACM TODS*, 24(4), Dec 1999.