XML Query Routing in Structured P2P Systems

Leonidas Fegaras, Weimin He, Gautam Das, and David Levine

University of Texas at Arlington, CSE
416 Yates Street, P.O. Box 19015, Arlington, TX 76019

{fegaras,weiminhe,gdas,levine}@cse.uta.edu

Abstract. This paper addresses the problem of data placement, indexing, and querying large XML data repositories distributed over an existing P2P service infrastructure. Our architecture scales gracefully to the network and data sizes, is fully distributed, fault tolerant and self-organizing, and handles complex queries efficiently, even those queries that use full-text search. Our framework for indexing distributed XML data is based on both meta-data information and textual content. We introduce a novel data synopsis structure to summarize text that correlates textual with positional information and increases query routing precision. Our processing framework maps an XML query with full-text search into a distributed program that migrates from peer to peer, collecting relevant document locations along the way. In addition, we introduce methods to handle network updates, such as node arrivals, departures, and failures. Finally, we report on a prototype implementation, which is used to validate the accuracy of our data synopses and to analyze the various costs involved in indexing XML data and answering queries.

1 Introduction

In the past few years, the P2P model has emerged as a new and popular computing model for many Internet applications, such as file sharing, collaborative computing, and instant messaging. A P2P network consists of a large number of nodes, called peers, that share data and resources with other peers on an equal basis. Peers are connected through a logical network topology implemented on top of an existing physical network, which may dynamically adapt to cope with peers joining and departing, as well as with network and peer failures. In contrast to traditional client-server architectures, a node in a P2P network can act as both a service provider and a client. Compared to traditional client-server systems, P2P systems are more scalable, flexible, fault-tolerant, and easier to deploy. Since no central coordination exists in a P2P system, there is no central point of failure. Additionally, network resources can be fully utilized and shared, and the server workload can be distributed among all the peers in the system.

Despite their benefits when compared to centralized systems, the most important challenge in designing a P2P system is search efficiency, especially in the presence of complex data and sophisticated search queries. Based on their search techniques, P2P systems can be classified into two main categories: unstructured

and structured P2P systems. In unstructured P2P systems, which include file sharing systems, such as Napster, Gnutella, KaZaA, and Freenet, there is no global protocol for data placement and the network topology is not tightly controlled. Their basic search strategy is to partially flood the P2P network with search messages or to randomly traverse the network until the desired data are found. Unstructured P2P systems offer easier deployment, more flexibility, and lower cost of maintenance than structured systems. However, they usually have poor search performance and do not scale very well, because the query load of each node grows linearly with the total number of queries, which in turn grows with the number of nodes in the system.

In a structured P2P system, the location of data is determined by some global scheme, such as a global hash function that hashes a search key to a node. Thus, keys are distributed to peers, forming a virtual Distributed Hash Table (DHT). By limiting the routing state of each peer to a small number of logical neighbors (typically logarithmic to the network size), structured P2P systems form overlay networks in which both the lookup time for a key and DHT maintenance take a logarithmic number of routing hops between peers. Thus, by adding an acceptable amount of lookup overhead, structured P2P systems offer higher availability and better scalability than traditional client-server architectures and unstructured P2P systems. Moreover, the well-distributed DHT-based data placement strategy naturally leads to load balancing in the system. Examples of DHT-based P2P systems are Pastry [9], Chord, and CAN.

While the benefits of structured P2P systems are significant, regardless of the type of data, there has been recent interest in indexing and querying data that are far more complex than the simple keys supported by DHT-based P2P networks, such as relational data [5, 4] and XML [3, 1]. The greatest challenges faced by these systems are data placement and query processing, because queries over these data are typically complex and, if not carefully optimized, may involve routing and processing massive data.

In this paper, we consider the problem of data placement, indexing, and querying large schema-less XML data repositories distributed over an existing P2P service infrastructure. Instead of developing a new special-purpose P2P architecture from the ground up, we are leveraging existing P2P technology, namely DHT-based P2P systems. We have chosen to work on XML because XML is now the language of choice for communication among cooperating systems. In our framework, any peer may publish XML documents, by making them available to all participating peers, and may submit queries against the published data. Although the published XML documents remain at the publication site, the P2P infrastructure serves as a distributed index for routing queries originated by any peer to the document sources that contain the query answers, rather than for retrieving the actual XML fragments of the answer. The query language considered by our framework is XPath, extended with simple syntactic constructs for full-text search. An XML document in our framework is indexed on both its textual content and its structural makeup, called the structural summary, which is a concise summary of all valid paths to data in the document.

Even though a formal schema, such as a DTD, would have been useful information for indexing and accessing data, our framework does not require it. The textual content of a document is summarized into data synopses, which capture both content and positional information in the form of bit matrices across the dimensions of indexed terms and their positions in the document. These matrices are small enough to accommodate frequent document publishing but precise enough to reduce the number of false positives in locating documents that satisfy the structural and content constraints of a query.

Although there are a number of earlier proposals on indexing and querying XML data distributed over a P2P network [3, 1], there is no work reported on complex XML query processing with full-text search on P2P networks that uses data synopses to selectively route queries to peers. The key contributions of our work can be summarized as follows:

1. We develop a framework for indexing XML documents based on summarized data (structural summaries and data synopses) and for mapping XML queries with full-text search into distributed programs that migrate from peer to peer, collecting relevant document references along the way.
2. We introduce novel data synopsis structures that have low data placement and maintenance overheads, and a high query routing precision. Our synopses correlate content with positional information, which results to a more accurate evaluation of textual and containment constraints in a query, when compared to regular Bloom filters. To the best of our knowledge, they are the first synopses to effectively address containment relationships between predicates in a query.
3. We introduce novel methods to handle network updates, such as node arrivals, departures, and failures.
4. Finally, we report on a prototype implementation to analyze the various costs involved in indexing XML documents and answering queries, and to validate our accuracy and scalability claims.

2 System Functionality

Although our framework is independent of the underlying DHT-based P2P architecture, our system is implemented on top of Pastry [9, 7]. A peer in our framework can make any number of its local XML documents public to the other peers through the process of *publishing*. Once published, an XML document is available to any peer for querying, until is removed by the document owner through the process of *unpublishing*. Our P2P network serves as a distributed index for routing queries to the peers who own the documents that can answer the queries. Document updates are done by unpublishing the entire document and publishing it again (as is done in most IR systems), rather than updating the distributed index to reflect the document updates.

Our main addition to the XPath syntax is the full-text search predicate $e \sim S$, where e is an arbitrary XPath expression, that returns true if at least one

element from the sequence returned by e matches the *search specification*, S. A search specification is an IR-style boolean keyword query that takes the form

$$\text{“term”} \mid S_1 \underline{\text{and}}\ S_2 \mid S_1 \underline{\text{or}}\ S_2 \mid (\ S\ )$$

where S, S_1, and S_2 are search specifications. A term is a keyword that must be present in the text of an element returned by the expression e. For example, the XPath query, QUERY:

```
//biblio//book[author/lastname ~ "Smith"][title ~ "XML" and "SAX"]/price
```

searches for books in biblio documents authored by Smith that contain the words “XML” and “SAX” in their titles and returns their prices. The result of the query is the set of addresses of all peers who own documents that satisfy the query.

3 Document Indexing

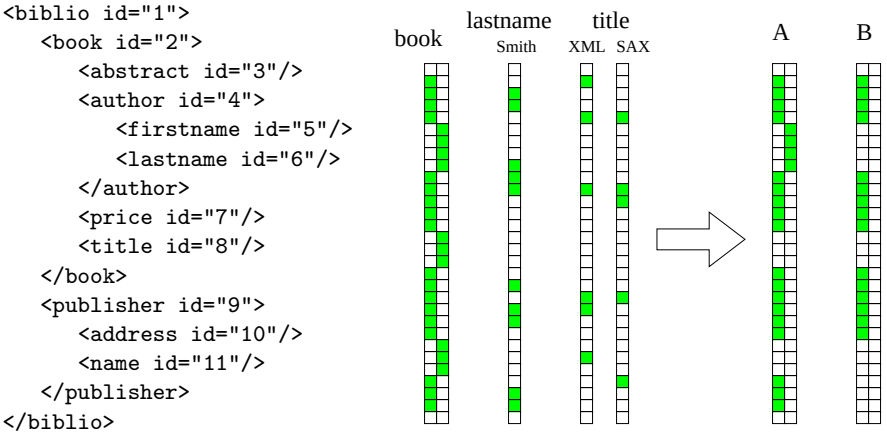

Fig. 1. A Structural Summary, and Testing QUERY Using Data Synopses

When published by a peer, the data of an XML document are indexed over the DHT of the P2P network on both the structural summary and the data synopses of the textual content of the document. A *label path* of an XML document is a simple path expression that contains child/attribute steps only and can distinguish a non-empty set of data nodes in the document. Each label path of a document is associated with a single node in the document's structural summary and with a single data synopsis. For example, using XML representation for convenience, one possible structural summary related to the bibliography documents searched by QUERY is shown at the left of Figure 1. In this figure, node 8 is associated with the label path /biblio/book/title. We use two types of data synopses: The structural summary nodes that contain text are associated with bit matrices, called *content synopses*, across the domains of text terms (keywords) and their positions in the document, while the internal structural

summary nodes are associated with bit vectors, called *positional filters*, across document positions. Both structural summary information and data synopses are distributed over the P2P network in a way that facilitates query evaluation, involving a small number of peers while achieving a good load balance. The DHT keys used for indexing a structural summary in a P2P network are the different tagnames in the summary, while the DHT key of a data synopsis is its label path. Given an XPath query, our framework is able to find all the plausible structural summaries applicable to the query using only one DHT lookup. In a typical DHT-based P2P system, this DHT lookup takes a number of routing hops logarithmic to the network size. A plausible structural summary is one that matches the *structural footprint* of the query, which is a path expression that captures all structural restrictions in the query.

The problem that we examine first is, given an XML document and an XPath query that contains search specifications, is the document likely to match the query? Since we are using synopses, our goal is to find an approximation method that reduces the likelihood of false positives, does not miss a solution, and does not require high placement and maintenance overheads. Since we are interested in evaluating IR-style search specifications, the document content is broken into simple terms (keywords), followed by stemming and stop word elimination. The resulting terms, called the indexed terms, are summarized in content synopses. More specifically, for each unique label path in a document that directly reaches text, we create one content synopsis that summarizes the indexed terms in this text along with their positions. The positional information is derived from the document order of the begin/end tags of the XML elements in the document. That is, the position of an element tag is the number of the begin and end tags that precede this tag in the document. The positional range of an XML element, on the other hand, consists of the positions of the begin/end tags of the element, while the positional range of an indexed term is the positional range of the enclosing element. That is, terms that belong to the same XML element have the same positional range. All positions in a document are scaled and mapped into a bit vector of size L, called a *positional bit vector*, so that the last position in the document is mapped to that last bit in the vector.

The positional dimension of the synopses is necessary due to the containment restrictions inherent in the search specifications of a query. For example, the search specification $e \sim$ "t_1" and "t_2" for two terms t_1 and t_2 becomes true if and only if there is at least one document node returned by e that contains both terms. Using one-dimensional term bitmaps alone, such as Bloom filters, and checking whether both the t_1 and t_2 bits are on, will give us a prohibitive number of false positives (as is shown in our experimental evaluation). For instance, using Bloom filters, our QUERY might have returned all documents that have one book whose author last name is Smith, a second book whose title contains XML, and a third book whose title contains SAX. Therefore, term position is crucial in increasing search precision and reducing false positives.

Given a document, the content synopsis H_p associated with a label path p is a mapping from an indexed term t to a positional bit vector. In our im-

plementation, a content synopsis is summarized into a bit matrix of fixed size $W \cdot L$, where W is the number of term buckets and L is the size of bit vectors. Then, $H_p[t]$ is the matrix column associated with the hash code of term t. If there are instances of two terms t_1 and t_2 in the document that have the same positional ranges (ie, they are contained in the same element), then the $H_p[t_1]$ and $H_p[t_2]$ bit vectors should intersect (ie, their bitwise anding should not be all-zeros). For example, we can test if a document matches the search specification title $\sim$ "XML" and "SAX" by bitwise anding the vectors H_8["XML"] and H_8["SAX"], which correspond to the node 8 (the book title) in Figure 1. If the result of the bitwise anding is all zeros, then the document does not satisfy the query (the opposite of course is not always true).

But given the bit vectors H_8["XML"], H_8["SAX"], and H_6["Smith"], how can we enforce the containment constraint in the QUERY that the book whose author last name is Smith must be the *same* book whose title contains XML and SAX? We cannot just use bitwise anding or oring. A first attempt is to use a positional filter F_p of size L associated with a label path p, so that for each XML element in the document reachable by the label path p, the bits in the bit vector that correspond to the element's positional range are all on. Unfortunately, this may result into bit overlaps of consecutive elements in the document, when their mapped positional ranges intersect. We address this problem by using M bit vectors F_p so that the positional range of the ith book goes to the i mod M bit vector. That way, two consecutive books in the document, the i and $i+1$ books, are placed to different bit vectors, thus reducing the likelihood of overlapping, which may in turn result to false positives. M should be at least 2, since it is very common to have a situation similar to that of consecutive books.

The basic operation in our framework to test element containment is called *Containment Filtering*, $CF(F, V)$, that takes a positional filter F and a bit vector V as input and returns a new positional filter F'. Function CF copies a continuous range of one-bits from F to F' if there is at least one position within this range in which the corresponding bit in V is one. For example, the right of Figure 1 shows how the data synopses of a document are used to determine whether this document is likely to satisfy QUERY (assuming $M = 2$). This is the case when at least one bit vector of B is not all zeros, where A is $CF(F_2, H_6$["Smith"]), B is $CF(A, and(H_8$["XML"], H_8["SAX"])), F_2 is the positional filter for node 2 (of books), and '*and*' is bitwise anding.

4 Data Placement and Query Processing

Our data synopses are used to route queries to peers who are likely to own documents that satisfy a query. To accomplish this, we introduce methods for placing and indexing data synopses over a P2P network and for locating documents relevant to a query based on these indexes. Our placement strategy for structural summaries is very simple: they are routed to peers using every distinct tagname from the structural summary as a routing key. Thus, locating all structural summaries that match the structural footprint of a query is accomplished by a single

DHT lookup by choosing a tagname uniformly at random from the query footprint as the routing key. A data synopsis is placed on the P2P network using its label path as the routing key. Since a label path may belong to multiple documents, all the relevant synopses from all these documents are placed at a single peer, the one whose Node Id is numerically closest to the Id of the label path. Thus, locating all document locations that satisfy the simple query $p \sim$ "term", for a label path p, is accomplished by a single DHT lookup by using the label path p as the routing key. Then, the document locations that satisfy this simple query are those whose content synopses, projected over "term", give a non-zero positional filter. To handle more complicated XPath queries, all the structural summaries that match the query footprint are extracted and all the distinct label paths that participate in the query's search specifications are collected and used as routing keys. Thus, the query is routed to the peers that contain the relevant data synopses, collecting and filtering document locations along the way. This is done over multiple documents at once. That is, the unit of communication between peers is a list of triples (peer,document,positional-filter), containing the owner and the id of a matching document along with the document positions that satisfy the query at the current point of query evaluation. This list is shorten at each visited peer by removing documents whose positional filters are all zeros. At the end, the query client collects all document locations that match the query and routes the query to the document owners for evaluation.

5 Handling Network Updates

There are three types of network updates that need to be handled by any P2P system: arrival, departure, and failure of nodes. While it is very important to maintain the integrity of the routing indexes, the integrity of data, which in our case are document references, is of secondary importance, as is apparent in web search engines that may return outdated links to documents. Both arrivals and departures can be handled without disrupting the query routing process. When a new node joins the overlay network and is ready to send and receive messages, it invokes the Pastry method notifyReady(). Our implementation of this method includes code that sends a message to the new node's successor to transport parts of its database to the new node. The node successor is derived from the node's Leaf set and is basically the immediate neighbor in the Id space with a larger Id. When the successor receives the message, it moves all structural summaries and data synopses whose routing Ids are less than or equal to the new node's Id to the new node. Therefore, the arrival of a new node requires two additional messages to transport data to the new node. When a node willingly leaves the overlay network, it routes all structural summaries and data synopses to its successor using one message only. The departing peer leaves the references to the local documents dangling and let the system remove them lazily.

A node failure is the most difficult network update to handle. When a peer P_1 receives a search request based on a tagname tag_1 to find all structural summaries that match a query footprint and does not find one, there are two possibilities:

either there was really no matching structural summary indexed in the DHT, or the predecessor node, who was closest to tag_1, had failed. Given that P_1 knows when its predecessor in the Id space fails (since, when this happens, it will receive a special message from Pastry), it can always distinguish the two cases: if the tagname Id is smaller than the failed predecessor Id, then it is the latter case. In that case, P_1 will choose another tagname tag_2 uniformly at random from the query footprint and relay the search request to another peer P_2 under the new key tag_2. In addition to the message relay, P_1 sends another message to P_2 asking to return all structural summaries associated with tag_1 to be published in P_1 (since P_1 now gets the requests for tag_1 keys). That way, the structural summaries of the failed peer are republished one-by-one lazily and on demand. Similarly, when a peer gets a request for a data synopsis based on the label path p and does not find one, and its predecessor had failed, it will check whether the Id from p is less than the Id of the failed predecessor. If so, it will abort the query and will scan the list of document publishers from the hit list routed along with query and will send a message to each publisher to publish the data synopses for path p again. Therefore, the restoring of data synopses associated with a failed peer in the P2P network is done lazily and on demand, and each time only one query has to be aborted.

6 Related Work

The closest work to ours is by Galanis *et al* [3] on XPath query routing in large P2P systems. Like our framework, the target of their distributed indexing is the location of data sources that contain the answer, rather than the actual XML fragments of the answer. Their structural summaries are similar to ours, since, for each tagname in an indexed document, they index all possible distinct paths that lead to this tagname in the document. This mapping is distributed in a DHT-based system using the tagname as the search key. Similarly, for each XML element that contains text, they store the text in the DHT index using the tagname of the XML element as the search key. This is contrary to our approach in which text is broken into terms before is indexed and label paths are used as keys. Unfortunately, the authors do not address the indexing cost, since their design is based on the assumption that querying is far more frequent than data placement. We believe that their framework is more suitable for data-centric XML data rather than to document-centric ones, since the latter may include large text portions inside specific tagnames, which results to the routing of large parts of a document to the same nodes. In their system, XPath queries are routed to peers based on the last tagname in the query, which serves as the DHT lookup key. Contrary to our approach, their evaluation of our example query will return even those nodes who own documents that have one book written by Smith and another with XML and SAX in their titles. That is, their method does not address containment relationships between predicates.

Another related framework is XP2P [1], which indexes XML data fragments in a P2P system based on their concrete paths that unambiguously identify the

fragments in the document (by using positional filters in the paths). The search key used for fragment indexing is the hash value of its path. Thus, XP2P can answer simple, but complete XPath queries (without predicates or descendant-of steps) very quickly, in one peer hop, using the actual query as the search key. The main drawback of this method is that retrieving a complete data fragment with all its descendants would require additional hops among peers by extending the query with the child tagnames of each retrieved fragment recursively, until all descendants are fetched. The descendant-of step requires even more searching by considering the parts of the query that do not have descendant-of steps and appending to them the child tagnames of the retrieved fragments (which makes it impossible to answer queries that start with a descendant-of step). More importantly, this technique has not been extended to include XPath predicates.

Although there are other proposed synopses and value distribution summaries for XML data, such as XSketch [8], their main use is in selectivity estimation, rather than in query routing in a P2P network. In [2], the structural summary (called Repository Guide) is served as a global schema that indicates how XML data are fragmented and distributed over the network. DBGlobe [6] exploits multi-level Bloom filters based on the structural summary of XML documents to route path queries to peers but they are based on structure information only. To the best of our knowledge, none of the synopses proposed by others summarizes both content with positional information in the same structure.

7 Performance Evaluation

We have built a prototype system to test our framework. It is built on top of Pastry [7] and uses Berkeley DB Java Edition as a lightweight storage manager. It is available at http://lambda.uta.edu/xqp/. The platform used for our experiments was a 3GHz Pentium 4 processor with 1GB memory on a PC running Linux. The simulation was done using Java (J2RE 1.5.0) with a 768MBs maximum memory allocation pool, from which 60% was allocated to the Berkeley DB cache. The experiments were performed over a cluster of 100, 1000, and 2000 peers in a simulated network on a single CPU. We used four datasets for our experiments, which were synthetically generated by the XMark and XMach benchmarks (see Figure 2).

	files	size	file size	tags	paths	msg/file	syn/file	syn size	msg/qry
XMark1	230	1.7 MBs	7.6 KBs	83	341	41.7	17.9	161 bytes	3.93
XMark2	2300	17 MBs	7.6 KBs	83	424	41.3	17.6	166 bytes	3.63
XMark3	11500	82 MBs	7.3 KBs	83	436	41.4	17.6	165 bytes	3.87
XMach	5000	87 MBs	17.8 KBs	1616	9206	28.9	17.7	547 bytes	3.52

Fig. 2. Benchmark Measurements

Our query workload consisted of 1000 random XPath queries generated from 50 files selected uniformly at random from each dataset. More specifically, each

selected file was read using a DOM parser and was traversed using a random walk, generating an XPath step at each node. When a text node was reached and a search specification was selected, search terms were picked from the text uniformly at random. That is, each generated query was constructed in such a way that it satisfied at least one document (the parsed document). Based on these four datasets and query workload, we derived the measurements in Figure 2, where msg/file is the average number of messages needed to publish one file from the dataset, syn/file is the average number of data synopses produced by each file in the dataset, syn size is the average size of an uncompressed synopsis, and msg/qry is the average number of messages needed to evaluate one query from the query workload. Note that these measurements are independent of the number of peers; they simply indicate the number of messages/synopses generated, rather than the number of distinct peers that receive these messages.

Note that the work by Galanis *et al* [3] does not capture positional information and does not address containment relationships between predicates, which, as we will see in our measurements, can considerably reduce the number of false positives. Furthermore, our indexing scheme is based on label paths (tagname sequences), which results to better load balancing than their scheme, which is based on single tagnames (the endpoints of a query). Nevertheless, we evaluated their system based on our datasets and query workload in terms of numbers of messages. Since their system was not available at the time of writing, we calculated the number of messages based on the analysis given in their paper. For XMark, they needed 17.6 msg/file, were each message had size 442 bytes, and 4.58 msg/qry. For XMach, these numbers were 17.7 msg/file, 1030 bytes, and 3.39 msg/qry. We can see that their system requires fewer messages for publishing than ours, although each publishing message is a little bit larger than ours on the average (since they have to publish the entire text). With our approach, in return, we gain better accuracy (as shown below) and better load balancing.

Based on the XMark datasets and query workload (1000 random queries), we measured the load distribution in our system for a network of 100, 1000, and 2000 peers. The results are shown in Figure 3. More specifically, for each one of the 3 XMark datasets, we grouped and counted the peers based on 1) the distribution of the number of messages needed to publish all the documents in a dataset; 2) the distribution of the number of content synopses after all the documents in a dataset have been published; 3) the distribution of the number of messages needed to evaluate 1000 randomly generated queries. These results can be easily explained given that the documents generated by XMark match a single DTD, resulting to a very small number of distinct tagnames and text label paths. For instance, from a network of 2000 peers, at most 436 peers are expected to receive all data synopsis placement/search requests, while the rest get none. For a network of 100 peers, though, the load is more evenly distributed. This load balancing skew, however, is not likely to happen with a heterogeneous dataset, when the number of distinct label paths is comparable to the network size. For example, the XMach dataset, which uses multiple DTDs, gives a better load distribution in processing 1000 randomly generated queries, as shown in

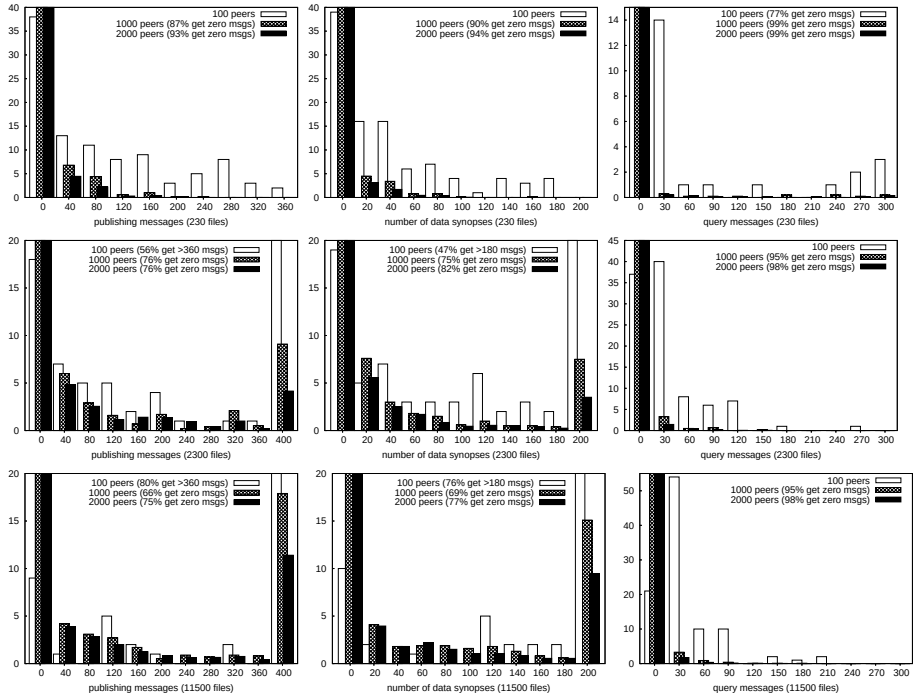

Fig. 3. Load Distribution based on the XMark Datasets (y Axis is % of Involved Peers)

Figure 4. More specifically, out of 2000 peers, 52.2% receive between 1 and 15 messages and 47.1% receive no messages (while for XMark3, 98% receive no messages, leaving the burden of query processing to 2%).

The second set of experiments was designed to measure the accuracy of data synopses. It was based on the XMark1 dataset on a single peer (since precision is not affected by the network size). The results are shown in Figure 4. For the first precision experiments, we used queries that match only one document from the dataset. The plot at the top right of Figure 4 shows the average number of false positives for various sizes of data synopses. Given a label path and a document, the width of its content synopsis is the number of document elements reachable by this path multiplied by the width factor. The height factor, when multiplied by the document size, gives the content synopsis heights. When the height factor was set to zero, then Bloom filters were used instead of content synopses (zero height). The size M is the height of positional filters. When $M = 0$, then no positional filters were used. We can see that, for random queries, the width factor affects precision more than the height factor (ideally, the number of false positives should be zero.) The plot at the bottom right of Figure 4 indicates that using Bloom filters (height factor = 0) yields twice as many false positives as when the height factor is ≥ 0.1.

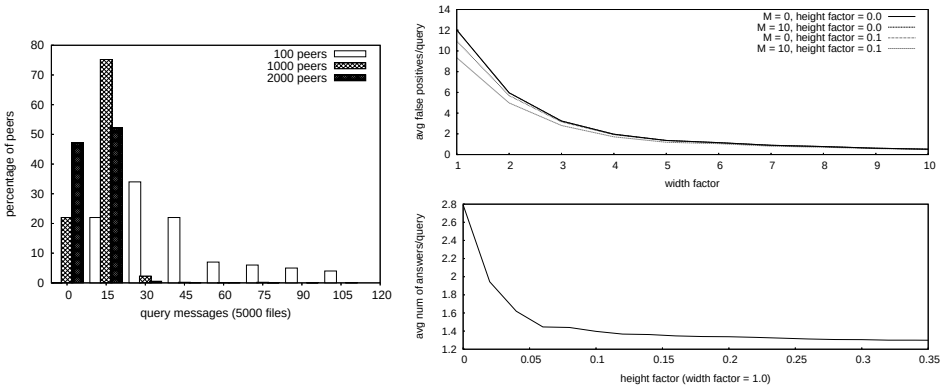

Fig. 4. Query Load Distribution for XMach and Data Synopsis Accuracy

8 Conclusion

We have presented a scalable architecture for indexing and querying XML data distributed over a DHT-based P2P system. The main contribution of our work is the development of a framework for indexing XML data based on the structural summary and data synopses of data, and for mapping an XML query with full-text search into a distributed program that migrates from peer to peer, collecting relevant data and evaluating parts of the query along the way. As a future work, we are planning to develop relevance ranking functions based on content synopses and use one of the known top-k threshold algorithms to reduce network traffic during querying.

References

1. A. Bonifati, *et al.* XPath Lookup Queries in P2P Networks. WIDM 2004.
2. J.-M. Bremer and M. Gertz. On Distributing XML Repositories. WebDB 2003.
3. L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt. Locating Data Sources in Large Distributed Systems. VLDB 2003.
4. A.Y. Halevy, Z.G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The Piazza Peer Data Management System. IEEE Trans. Knowl. Data Eng. 16(7): 787-798 (2004).
5. R. Huebsch, *et al.* The Architecture of PIER: an Internet-Scale Query Processor. CIDR 2005.
6. G. Koloniari and E. Pitoura. Content-Based Routing of Path Queries in Peer-to-Peer Systems. EDBT 2004.
7. Pastry. http://freepastry.rice.edu/.
8. N. Polyzotis and M. Garofalakis. Structure and Value Synopses for XML Data Graphs. VLDB 2002.
9. A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. International Conference on Distributed Systems Platforms 2001.