Path Query Routing in Unstructured Peer-to-Peer Networks

Nicolas Bonnel, Gildas Ménier, and Pierre-Francois Marteau

Valoria, Université de Bretagne-Sud
Campus de Tohannic, Bat. Yves Coppens
56000 VANNES
{nicolas.bonnel, gildas.menier, pierre-francois.marteau}@univ-ubs.fr

Abstract. In this article, we introduce a way to distribute an index database of XML documents on an unstructured peer-to-peer network with a flat topology (i.e. with no super-peer). We then show how to perform content path query routing in such networks. Nodes in the network maintain a set of Multi Level Bloom Filters that summarises structural properties of XML documents. They propagate part of this information to their neighbor nodes, allowing efficient path query routing in the peer-to-peer network, as shown by the evaluation tests presented.

1 Introduction

There is a growing need for large databases of semi-structured documents and this raises management and querying problems that are specific to this kind of data. XML [1] is a standard for semi-structured document encoding. Building an index database allows to speed up the querying process [2] and in general this database is larger than the database itself. Distributing the index database allows to store and manage a larger amount of information.

Peer-to-peer (P2P) systems can be used to manage XML data [3]. They can have different architectures. Napster [4] for instance is a centralized P2P network that was very popular in the early 00'. The use of a central repository to answer queries makes this system poorly scalable and vulnerable to failure. Others P2P systems don't rely on a central server : they are decentralized, thus they are very scalable, and fault tolerant. They are divided in two categories.

First, structured P2P networks link network topology and location of data. Most of them implement a Distributed Hashtable (DHT) [5–9] and provide one basic operation : given a key, they map the key to a node. This is performed by using a distributed hash function. They use content routing to forward the key to the corresponding node. They are very well suited to retrieve rare information (i.e. with a low number of replicas). Their main limitation is that it is very costly to perform range and approximative queries, because hashing destroys the order on keys. Data clustering on those systems is mostly delicate, as data placement is given by the hash value of keys.

Second, unstructured P2P networks have no constraint between location of data and network topology. Gnutella [10] is an example of such working system. Query forwarding can be achieved either by flooding [11] - consuming a lot of bandwidth - or by random walk : a path is selected randomly according to a uniform distribution. They are suited to retrieve highly replicated data, but have limitations for rare information retrieval. Data clustering on such systems can be achieved because there is no constraint on data placement. Nowadays, most of the P2P systems used have a decentralized unstructured topology.

Our current work focuses on indexing XML data and distributing the index database. As we need to be able to perform approximative path queries, we choose an unstructured P2P architecture. The primary purpose of the nodes on the network is not to manage this index database. They all may have to set aside the management of the database if a user would need the resources on this node for his own purpose. For this reason our P2P network don't have super peers. The use of an unstructured P2P architecture allows us to have a network topology as close as possible to the physical topology and can lead to use less network resources. As we need to be able to perform path queries from all nodes of the network, we need a mecanism to route path queries. In this article, we describe a scheme based on the use of exponentially decaying multi level Bloom filters to address this problem.

Section 2 recalls previous work, section 3 presents our system, section 4 shows our experiments and section 5 talks about possible future work.

2 Related Work

2.1 Unstructured P2P routing mechanisms

Many studies have already dealt with this problem. For instance, the problem of routing path queries in unstructured peer to peer network has been tackled by [12–14]. However they all consider hierarchical topologies with super-peers, which we want to avoid for the reasons stated previously.

A hybrid search mechanism has been proposed [15] to achieve routing for both common and rare items. This system uses flooding to locate highly replicated information and a DHT to locate rare information.

Random walk efficiency can be improved while launching k similar queries. k-random walk has been proposed by [11] that improves query response rate with little network trafic.

Distributed-index mechanisms can improve the efficiency of content routing. Different Routing Indices (RI) have been proposed by [16] : compound, hop-count and exponential. Various kinds of Bloom filters [17] have been used to encode these distributed indexes.

Attenuated Bloom filters have been proposed by [18]. Exponentialy Decaying Bloom Filter (EDBF) has been proposed by [19] and Multi-level Bloom Filter (MLBF) has been introduced by [12]. Both are presented later in this article.

2.2 Bloom filters

A Bloom filter [17] is a data structure that answers approximatively to set membership queries. It is made of an array of m bits and k hash functions $h_1, ..., h_k$, as shown in figure 1.

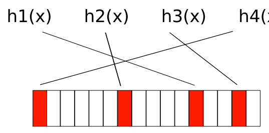


Fig. 1. Insertion of an element x in a Bloom filter of length m=16 and k=4 hash functions.

When no element has been inserted into the filter, all bits are set to 0. When an element x is inserted into the filter, all bits given by the k hash functions $h_i(x)$ are set to 1 with a classical bitwise-or operation. A test membership of an element to a set is answered by checking that all the bits given by the k hash functions are set to 1.

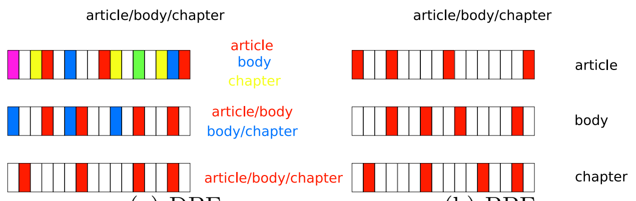
False positives are possible but false negatives are not. The probability of having a positive answer for an element not belonging to the filter, with n being the number of elements inserted in the filter, is $(1-(1-\frac{1}{m})^{nk})^k$.

More elaborated Bloom filters have been proposed [18, 20–22]. For instance Counting Bloom Filter (CBF) [23] allows the removing of elements from the filter. In those filters, the array of bits is replaced by an array of integer, each of them acting as a counter. Inserting an element is performed by increasing integers given by the k hash functions. Removing an element is performed by decreasing those integers. A set membership query is answered by checking that all integers are strictly positives.

Exponentialy Decaying Bloom Filter (EDBF). This filter has been proposed by [19]. Introducing an element is performed as in a classical Bloom filter. However, querying for an element x gives the number $\theta(x)$ of bits equal to 1. Thoses filters are then used to encode probabilistic routing tables, in which $\frac{\theta(x)}{k}$ is the probability to find element x among a given link in the network (Each node maintains a filter for each neighbor he has). This probability decays exponentially with the number of *hops* (or node transitions) from the node where the element x is stored.

Nodes update their filters periodically. The filter for one neighbor is updated with the information attenuated from all other neighbors and the information without attenuation from the local EDBF of the node. The attenuation is performed by resetting each bit to 0 with a probability $1/d$, where d is the decay of the filter. The aggregation of information corresponds to a bitwise-or operation.

Multi Level Bloom Filters (MLBF). It has been introduced by [12]. The main idea they propose is to use a set of Bloom filters to describe structural properties of set of XML documents. There are 2 sets of Bloom filters : Breadth Bloom Filters (BBF) and Depth Bloom Filters (DBF), as shown in figure 2. A BBF is composed of i Bloom filters $\{BBF_1, ..., BBF_i\}$. Inserting an XML document is performed by inserting all nodes of level l in BBF_l. A DBF is composed of j Bloom filters $\{DBF_1, ..., DBF_j\}$. Inserting an XML document is performed by inserting all subpath of length l in DBF_l. According to the authors, BBF works better than DBF in general case, but the last can drive path queries with ancestor-descendant relationships.



(a) DBF (b) BBF

Fig. 2. Multi level Bloom filter with a BBF and DBF of size 3. Each subfilter has a length m=16 and k=4 hash functions. This figure illustrates the insertion of the path "article/body/chapter" in both filters.

To perform routing, nodes are then clustered in a hierarchical organization. A set of nodes are designated as *root nodes*, according to their storage and processing capabilities, and are connected to a main channel that allows them to communicate between themselves. Each root node has a merged filter that contains all the elements in its children peers and a local filter that contains elements hosted on this node. When a root node receives a query, it first checks its local filter. It then propagates the query to its children if there is a hit in its merged filter.

A hierarchical organization is adapted to a network of nodes with different processing and storage capabilities. However, our system is designed to work on a network in which nodes will have varying processing and storage capabilities. For this reason, we can't have a hierarchical topology and need a mechanism to drive path queries in a random graph like topology. EDBF have proved to be efficient in such topologies but can't drive path queries. Next section presents our system that perform path query routing in a random graph like network topology using an exponentially decaying version of MLBF.

3 System design

3.1 Path indexing scheme

Each node of the network performs the indexing of a part of the set of XML documents. For each XML document, paths leading to content are indexed in a hashtable, using the QDBM library [24].

Each node carries a BBF of size l and a Reverse Breadth Bloom Filter (RBBF) composed also of l Bloom filters $\{RBBF_1, ..., RBBF_l\}$ of size l. For each path P of length k, $P = /e_1/.../e_k/$ and $\forall i \in [1, \ldots, k]$ we insert e_i in BBF_i and $RBBF_{k-i}$. For each node, BBF and RBBF are made of CBF, so that elements can easily be removed.

We don't use DBF for two reasons. First, according to [12], they are less efficient than BBF. Second, they cannot be used efficiently to drive path queries containing unknown elements. For instance, let us consider a query such as /A/?/C/. This query can be driven using information in the first and third level of the BBF, whereas only information in the first level of the DBF (paths of length 1) can be used.

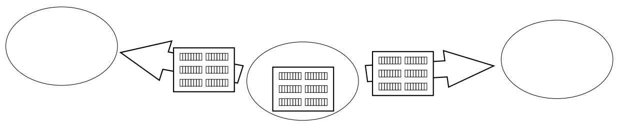


Fig. 3. Example of filter settings on a node with 2 neighbors. BBF and RBBF have a size of 3

We encode probabilistic routing table using an exponentially decaying version of BBF and RBBF. Thus, each node has two filters for each neighbor, as shown in figure 3. Updating the filters is performed as described in [19].

3.2 Querying mechanism

The query language we consider is a subset of the XML Path Language (XPath) [25]. Let E be the set of all XML element names. Let "?" be a don't care element and $E^? = E \cup \{"?"\}$. Let S be the set of relationship symbols, $S = \{"/", "//"\}$. "/" describes a parent-child relationship, and "//" describes an ancestor-descendant relationship. "/" is a particular case of "//". A path query P of length k is defined as a word on $E^? \cup S$, with $P = s_0\ e_1\ s_1\ ...\ e_k\ s_k$, where $e_i \in E^?$ and $s_i \in S$. $/e_1$ stands for the root element. We assume each local node has a mechanism to answer queries exactly. Either the begining or the end of the searched paths must be known, thus s_0 and s_k cannot be at the same time both "//".

Let P be a path query containing no "//" relationship, $P = /e_1/.../e_k/$. Let $Q(F, x)$ be the query response of the presence of the element x in the filter F. Querying MLBF is performed as follows :

$$Q(BBF, P) = Q(BBF_1, e_1).Q(BBF_2, e_2) \dots Q(BBF_k, e_k) \quad (1)$$

$$Q(RBBF, P) = Q(RBBF_1, e_k).Q(RBBF_2, e_{k-1}) \dots Q(BBF_k, e_1) \quad (2)$$

Querying counting filters return *true* or *false* (exact query), and the "." in equations 1 and 2 stands for the logical AND. Querying exponentially decaying filters return a result in $[0, 1]$ (approximative query), and the "." in equations 1 and 2 stands for the product.

MLBF querying is performed by querying BBF and RBBF. If the path query contains "//", the subpath before the first "//" is answered by BBF and the subpath after the last "//" is answered by RBBF. If the path query contains only parent-child relationships, the whole query is answered by both filters. The global result is then given by the product (exponentially decaying filters) or the logical AND (counting filters) of the results given by BBF and RBBF.

3.3 Clustering

Clustering similar data increases routing efficiency for two reasons. First, it allows to speed up approximative and range queries. As similar information is located on neighbor nodes, it reduces communication and minimizes resources requirement. Second, it decreases the number of bits set to 1 in Bloom filters. This leads to a better discrimination between filters and increases gradient routing performance.

There are two ways to perform clustering : node and data clustering. The former aggregates nodes with similar content, while the latter aggregates data that are closed on the same node. As we would like to have as few constraints as possible on nodes, and especially on the network topology, we perform data clustering. Our algorithm clusters similar XML documents on the same nodes or on neighbor nodes.

Let $P = /e_1/.../e_k/$ a path of length k. We define for the node N :

$$\begin{aligned} \psi(N, P) &= |\{i \in [1, \dots, k] / BBF_i(e_i) = true\}| \\ &\quad + |\{i \in [1, \dots, k] / RBBF_{k-i}(e_i) = true\}| \\ \omega(N, P) &= \psi(N, P) + \sum_{x \in neighborhood\ N} \psi(x, P) \\ S(N) &: \text{ a linear function of the space available on node N} \\ D_N &: \text{ the set of XML paths stored in N} \end{aligned}$$

Our clustering algorithm works as follows : each node N launches periodically an agent. This agent contains a copy of an XML path P taken at random from D_N and the indexing information related to this path, a Time To Live (TTL), a reference to the node N, and the score of the node N for this path P. This score is given by the product $\omega(N, P).S(N)$.

The agent moves to a random neighbor until his TTL reaches 0 or until it finds a node with a better score for the path P. He keeps track of the sequence of visited nodes. If a better node is found, the path P and related indexing information is moved from the agent to that node, and then a deletion message for this path and the associated information is sent to the original node N using routing information stored by the agent. If the node already stores the path P, indexing information is merged. If the TTL reaches 0 the agent is discarded, and the path P remains on the node N.

3.4 Query forwarding

Query forwarding is performed using the Scalable Query Routing algorithm (SQR) proposed by [19]. If the query is forwarded for the first time from a node, it is sent to the neighbor with the highest score when querying the exponentially decaying MLBF of the link to this neighbor. If the query has already been seen, it is forwarded to a random neighbor.

As we only use Breadth Bloom Filters, there is no relation between the elements that compose a path. For instance, if the paths /book/chapter/ and /article/abstract/ are inserted into the MLBF, it will answer true for the paths /book/abstract/ and /article/chapter/, even if they are not in the filter. However, because of our clustering algorithm, if such paths exist, they will likely be in the same node, or in the neighborhood of the node.

4 Experiments

Experiments have been made with a simulation written in java. The settings of these experiments can be found in table 4. In those experiments, information was not replicated. A subset of the wikipedia collection from INEX 2006 [26, 27] containing 260000 XML documents has been used. We used a stoplist containing commons elements, such as ¡article¿, and small length elements, like ¡b¿ used for bold text. We removed all those elements in the paths we indexed. For these experiments, we made comparisons with random walk rather than with flooding, because it would be too costly to retrieve information located far away from the query source.

Figure 5 show the evolution of the number of paths moved, the number of indexed paths and the filters occupation. Merging similar paths on the same nodes allows to lower the number of bits set to 1 in filters by a factor of 3. Our algorithm converge to a minimum, as the number of paths moved gradually get closer to 0. The comparison of routing efficiency for path queries between SQR and random walk has been performed by averaging 1000 different queries launched 20 time each. We used an hop count measure instead of a TTL one because the computer used could have varying CPU or memory resources (for these experiments).

Parameter	Value
Number of nodes	200
Node degree	3 - 8
XML documents per node	1300
Length of filters	2^{13}
Size of BBF and RBBF	3
Number of hash functions	32
Decay	8

Fig. 4. Experiment settings

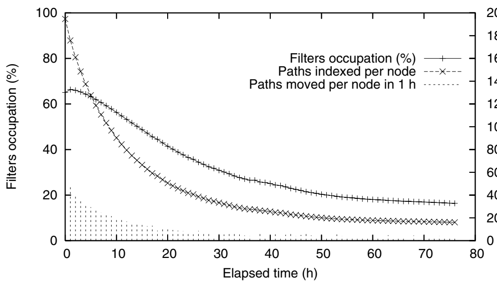
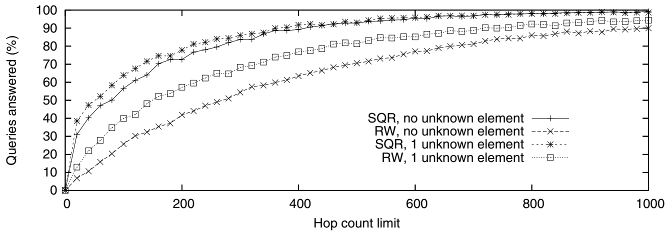
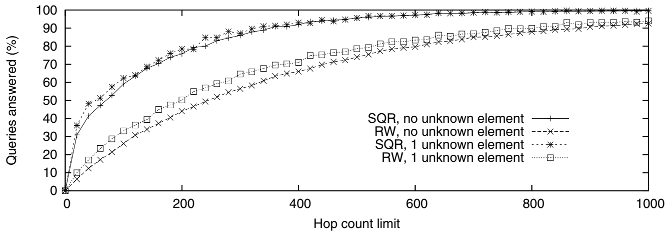


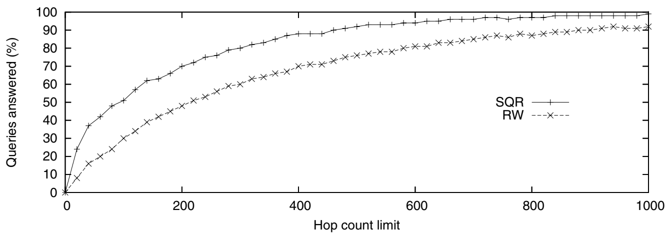
Fig. 5. Impact of clustering.



(a) Path queries with 2 elements.



(b) Path queries with 3 elements.



(c) Path queries with an ancestor-descendant relationship between 2 elements.

Fig. 6. Comparison of routing efficiency between SQR and random walk.

Figure 6(a) shows the comparison of routing efficiency for 2 elements path queries and figure 6(b) for 3 elements path queries. There is a difference between the 2 random walk measures because path queries with 1 unknown element (for instance */document/?/*) cover more paths than path queries with no unknown element (for instance */document/book/*). However, as the hop count limit increase, the 2 SQR measure get closer, as having no unknown element provide more information for routing.

Figure 6(c) shows results with 2 elements path queries, linked by an ancestor-descendant relationship (for instance */document//paragraph/*). Although performances are a bit lower than those with 2 elements path queries containing only parent-child relationships, there still a good increase of routing efficiency. For instance, 70% of queries can be answered with a *hop* count limit two times lower than with random walk.

5 Conclusion and future work

We have introduced a way to perform stochastic approximative path query routing in an unstructured P2P network. We have shown that with our clustering algorithm, SQR outperforms random walk to forward simple approximative path queries. Filters used for routing can be maintained at low cost, as we can control the frequency of the updates, and because Bloom filters use few memory.

Further research efforts are required to be able to drive more elaborated path queries. Furthermore in our experiments information was not replicated. We will study the impact of information replication to see how SQR performs with path query through multiple data replica. For the time being, our simulator works on a single computer. We plan to distribute our application on a cluster to be able to perform testing on much larger XML databases.

References

1. Bray, T., Paoli, J., Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0 - W3C recommendation 10-february-1998. Technical Report REC-xml-19980210 (1998)
2. McHugh, J., Widom, J., Abiteboul, S., Luo, Q., Rajamaran, A.: Indexing semistructured data (1998)
3. Koloniari, G., Pitoura, E.: Peer-to-peer management of xml data: issues and research challenges. SIGMOD Rec. **34**(2) (2005) 6–17
4. Napster: Napster homepage (2001) http://www.napster.com/.
5. Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In: ACM SIGCOMM 2001, San Diego, CA (September 2001)
6. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA (2000)
7. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science **2218** (2001)

8. Zhao, B.Y., Kubiatowicz, J.D., Joseph, A.D.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley (April 2001)
9. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A distributed anonymous information storage and retrieval system. Lecture Notes in Computer Science **2009** (2001)
10. Clip2: The gnutella protocol specification v0.4 (2002)
11. LV, C., CAO, P., COHEN, E., LI, K., SHENKER, S.: Search and replication in unstructured peer-to-peer networks (2001)
12. Koloniari, G., Pitoura, E.: Content-based routing of path queries in peer-to-peer systems. In: Proceedings of the EDBT'04 International Conference, Heraklion, Crete, Greece. (2004)
13. Dragan, F., Gardarin, G., Yeh, L.: Mediapeer: A safe, scalable p2p architecture for xml query processing. In: DEXA Workshops. (2005) 368–373
14. Wang, Q., Jha, A.K., Ozsu, M.T.: An xml routing synopsis for unstructured p2p networks. waimw **0** (2006) 23
15. Loo, B.T., Huebsch, R., Stoica, I., Hellerstein, J.M.: The case for a hybrid p2p search infrastructure (2004)
16. Crespo, A., Garcia-Molina, H.: Routing indices for peer-to-peer systems (2002)
17. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13**(7) (1970) 422–426
18. Rhea, S.C., Kubiatowicz, J.: Probabilistic location and routing. In: Proceedings of INFOCOM 2002. (2002)
19. Kumar, A., Xu, J., Zegura, E.W.: Efficient and scalable query routing for unstructured peer-to-peer networks. In: Proc. of IEEE Infocom. (2005)
20. Cohen, S., Matias, Y.: Spectral bloom filters. In: SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM Press (2003) 241–252
21. Chazelle, B., Kilian, J., Rubinfeld, R., Tal, A.: The Bloomier filter: An efficient data structure for static support lookup tables
22. Kumar, A., Xu, J., Wang, J., Spatscheck, O., Li, L.: Space-code bloom filter for efficient per-flow traffic measurement. In: Infocom, 2004. (2004)
23. Li, F., Pei, C., Jussara, A., Andrei, B.: Summary cache: A scalable wide-area web cache sharing protocol. In: Proceedings of SIGCOMM'98, Computer Sciences Department, Univ. of Wisconsin-Madison (February 1998) Technical Report 1361.
24. Hirabayashi, M.: Quick database manager (2006) http://qdbm.sourceforge.net/.
25. Robie, J., Fernández, M.F., Boag, S., Chamberlin, D., Berglund, A., Kay, M., Siméon, J.: XML path language (XPath) 2.0. W3C proposed reccommendation, W3C (November 2006) http://www.w3.org/TR/2006/PR-xpath20-20061121/.
26. INEX: Initiative for the evaluation of xml retrieval (2006) http://inex.is.informatik.uni-duisburg.de/2006/.
27. Denoyer, L., Gallinari, P.: The wikipedia xml corpus. SIGIR Forum **40**(1) (2006) 64–69