DISTRIBUTED BLOOM FILTER FOR LOCATING XML TEXTUAL RESOURCES IN A P2P NETWORK

Clément Jamard, Laurent Yeh and Georges Gardarin
PRiSM Laboratory, University of Versailles, 45 av. des Etat-Unis, 78000, Versailles, France

Keywords: XML, XQuery Text, P2P Network, Database System, Bloom Filter, Indexation.

Abstract: Nowadays P2P information systems are considered as large scale distributed databases where all peers can provide and query data in the network. The main challenge remains locating relevant resources. In the case of XML documents, keywords and structures must be indexed. However, the major problem for maintaining indexes of huge textual XML documents is the cost for connecting/disconnecting: indexing a quantity of keys requires the transit of many messages in the network. To reduce this cost we adapt the Bloom Filter principle to summarize peer content. Our Bloom Filter summarizes both structure and value of XML document and is used to locate resources in a P2P network. Our originality is to propose techniques to distribute the Bloom Filter by splitting it into segments using a DHT network. The system is scalable and reduce drastically the number of network messages for indexing data, maintaining the index and locating resources.

1 INTRODUCTION

Peer-to-Peer (P2P) network is a technology used to share and query information in a distributed manner. P2P networks offer dynamicity of data sources, robustness, scalability, reliability and no central administration. With the emergence of XML as a standard for representing and exchanging data, P2P networks have been adapted to share structured information. Coupling P2P networks with XML databases could emerge new query possibilities at world scale. P2P systems are used to locate relevant sources by indexing data that peers intend to share on other peers. Queries are then resolved in a distributed manner.

In P2P network few attention has been paid for peer dynamicity (connecting, disconnecting or updating peer) and for indexing XML with massive text data. In fact, peer dynamicity and XML data indexing imply heavy traffic by sending every key to index, that is a bottleneck for the system. Without dynamicity, or with an heavy cost for each action like peer connection, the P2P network appears more static than dynamic that is in opposition with the P2P paradigm.

Two kinds of P2P systems are focused on locating XML sources. First, P2P systems like SomeWhere (Rousset et al., 2006) or Piazza (Halevy et al., 2003) are focused on locating efficiently data sources using mappings between peers. An advantage of this approach is that it does not require to index data in the network but connecting a peer in those networks requires constructing complex mappings, at an heavy cost. Second, many P2P networks are built over robust and scalable DHT (*Distributed Hash Table*) methods ((Rowstron and Druschel, 2001), (Stoica et al., 2001), (Ratnasamy et al., 2001)). In KaDoP (Abiteboul et al., 2005), XML documents are decomposed into elements (node, text value) that are indexed in the DHT. For each item, a message is sent in the network for indexing the value. Many messages are sent in the network as XML documents are composed of several elements. Pathfinder (Gardarin et al., 2006) is an alternative to index structured information; entries of the index are organized as a compressed sequence of elements and values. Although this approach compresses the required index size and speed up twig queries, the number of entries to be shipped in the network remains huge.

DBGlobe (Koloniari et al., 2003) proposes an alternative to these P2P systems by using Bloom Filter (Bloom, 1970) for locating resources in a P2P network. A Bloom Filter is a compact data structure used for testing the membership of an element in a set. A Bloom Filter is a bit array of size *m* initially set to 0 associated to *k* hash functions. Each function maps a key value to a bit array entry. For checking if the filter accepts a given word *w*, all relevant array entries determined by the *k* functions must be set to 1. Notice that a value succeeding the test might not be in the document. This phenomenon is called a *false positive answer*.

In DBGlobe, a Bloom Filter is only used to summarize XML document structure. Peers are connected by group on a P2P bus network, where each group provides a filter summarizing participant structures. This filter is used to orient query to relevant group of peers. The cost for finding relevant peers is important as the query must traverse several group of peers before accessing the relevant data; the network traffic can become a bottleneck and it only indexes structure.

We propose Distributed Bloom Filters to filter XML content and structure. Our data model allows to distribute a Bloom Filter on a DHT network by splitting it into segments. It behaves like a non dense index; keys are indexed with a probability to find in the network. The work is focused on minimizing communications and data exchanged for peer connection, disconnection or querying data. We also propose a method to control the probability of false positive for efficiency of the solution.

The paper is organized as follows. In Section 2 we define the model of Distributed Bloom Filter. We describe in Section 3 peers behavior for locating resources using Distributed Bloom Filters in a DHT-based network. In Section 4 we present experimental evaluation that show the benefit and efficiency of our method. In Section 5, we concludes our work.

2 DISTRIBUTED BLOOM FILTER

A Distributed Bloom Filter (called DBF) is a structure adapted from Bloom Filter to summarize XML data on content and structure. In this section we describe how to summarize XML data using Bloom Filters. Then we present a method to distribute and query efficiently it on a DHT network. Finally we propose a technique to control the probability of false positive to guarantee good performance.

2.1 Content and Structure Filter

We use Bloom Filter to check the presence of a structural expression correlated with a value in peers data. Keys inserted in the filter are composed of a path and

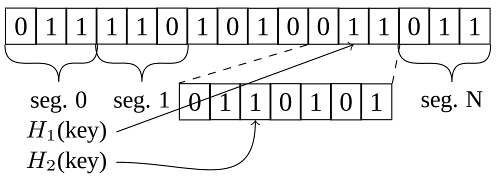


Figure 1: Distributed Bloom Filter and Shadow Segment.

a value. Every XML document can be mapped to a set of *value-localization-paths* VLP = $\{vlp_i\}$ where vlp_i is of type: $/a_1/.../a_i[V]$. Only the node a_i contains the value V. When an element node contains several words, it could be split into a set of value-localization-paths. For example, `<Book><Title> XML Introduction </Title></Book>` produces two value-localization-paths `/Book/Title[XML]` and `/Book/Title[Introduction]`.

The *k* H_i functions are used to determine which array entries of the filter to set to 1. Each function captures both structure and value information; H_i function is composed of H_{pi}, a path coding function, and H_{vi}, a value coding function:

$$H_i(key) = H_{pi}(key.path) * H_{vi}(key.value)$$

The H_{vi} function is a typical hash function with value range from 0 to $(m-1)$, where m is the size of the array. The H_{pi} function uses a path encoding technique described in (Jagadish et al., 2005). The main idea is to map the path domain to a range of value between 0 and 1. Each unique path is mapped to a unique value. More details can be found in (Jagadish et al., 2005). The H_i function sets a bit to 1 in the Bloom Filter array, with range from 0 to $(m-1)$. Thus, two different value-localization-paths with same value (e.g., `/Book/Title[XML]` and `/Article/Title[XML]`) will set to 1 different entries of the Bloom Filter.

2.2 Distributing a Bloom Filter

To distribute the filter, the bit array is split into segments of equal size. Segments are distributed over peers in a DHT network using the primitive `put(key, value)`; the segment number is used as a routing key. This splitting method implies two major problems:

- each peer responsible of a segment number will receive segments from every peers in the network,
- checking each of the k hash functions requires to contact many peers as the set of H_i function could cover several segments.

To distribute more evenly segments, we introduce the notion of document theme. Each document that a peer intends to publish or query is associated to a theme. A user can find the relevant themes from a catalog of all existing predefined themes shared in the network. The theme is combined with the segment number to determine the key used for the `put(key,value)`. Therefore, the i^{th} segments of the DBF are not indexed on a same peer as they belong to different themes.

To avoid checking several segments, we constraint H_2 to H_k functions to cover a single segment. Consequently, the $H_1(key)$ function plays two roles: filtering and routing purpose. It filters values as a traditional hash function, and it is also used to determine the segment number where other hash functions are constrained. The segment number ($m \div H_1(key)$, m being the DBF size), combined with the theme, is used as the key of the primitive `put(key, value)` for determining the peer storing the segment to check.

2.3 Controlling Segment Selectivity

The selectivity of a Bloom Filter depends on the size m of the filter, the number k of functions, and the number n of keys inserted. The probability of having false positive answers (i.e., the probability of having the *k* positions set to 1 for an element not in the set) is given by $(1 - e^{-kn/m})^k$. As the number *k* of function is fixed, the probability depends on the ratio *n/m*.

The false positive probability may imply a lot of useless network communications. In fact, each time a value is successfully filtered, the peer that created this filter is contacted. Therefore, controlling the false positive probability by keeping it below a threshold is important as it will reduce network traffic.

When a source peer adds, removes, or modifies documents, its DBF must reflect the changes. Some techniques (Fan et al., 2000) are available to support removal in Bloom Filter. Our goal is to maintain under a threshold the selectivity of a Bloom Filter after insertions.For controlling the selectivity, we introduce shadow segments. When the ratio *n/m* makes the probability exceed the threshold, a shadow segment with an augmented size is used. Each segment keeps the number of keys inserted so far. The shadow segment size is computed so that the ratio *n/m* keeps the probability under the threshold. When a shadow segment is created, keys have to be rehashed in the shadow segment using the new hash functions. The H_1 function remains the same, determining the segment number, and others H_i functions range is modified to cover the shadow segment interval. With this approach we can adjust the size of a bloom filter dynamically according to the required need.

3 LOCATING DATA SOURCES

3.1 Network Architecture

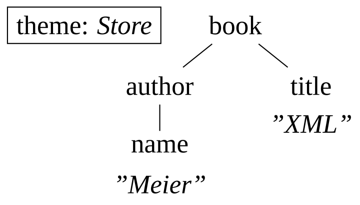
As in traditional P2P networks, a peer can be a *client*, a *server*, or a *router*. We add a fourth role: a peer is also a *controller* for managing segments of DBF. The client role is used for querying the network. A server peer shares data on the network. For a server peer, the DBF created from its data is split into segments; segments are distributed through the network using the DHT `put(key, value)` function for sending the segment to a controller peer. The message sent through the network contains: *(i)* The segment of the distributed Bloom Filter. *(ii)* A set of Bloom Filter hashing functions ($H_2(key)...H_n(key)$). *(iii)* The IP address of the server peer. Each peer is a router, routing messages according to the DHT principles. A controller peer manages distributed segments of others peers. The role of a controller peer is to check managed segments according to the Bloom Filter principles.

3.2 Locating Relevant Sources

Queries processed in our system are simple content and structure queries with absolute path expressions (i.e. only child axis). A query can be expressed as a tree of path expression where keywords are attached to leaf nodes, and a theme. A query tree is decomposed into value-localization-paths, as an XML document. Each value-localization-path is inserted in a demand message, used to resolve in a distributed manner the query. A demand, illustrated at bottom of figure 2, is organized as follows:

- a `step` attribute indicating the current process: checking the DBF (*checkingDBF*) or contacting server peers (*checkingSRC*),
- a `from` attribute for the client peer address,
- `vlp` elements representing value-localization-paths of the query. It stores the `theme` of the query, the `path` and the `value` to search. The `state` attribute indicate wether it has been resolved on a controller peer (*found*) or in instance to be (*looking*),
- a `results` element storing source peers filtered by DBFs.

Client Peer At creation time, a new demand message is created with step set to *checkingDBF*. The algorithm 1 describes the behaviours at the client peer. First, from the query tree, a set of value-localization-paths is extracted. The value-localization-paths (vlps)



```
<Dem step="checkingDBF" from="193.51.25.187">
  <vlp state="looking" theme="Store">
    <path>/Book/Author/Name</path>
    <value>Meier</value>
  </vlp>
  <vlp state="looking" theme="Store">
    <path>/Book/Title</path>
    <value>XML</value>
  </vlp>
  <results/>
</Dem>
```

Figure 2: Query tree and routing demand.

are ordered in the demand according to their value issued from the H_1 function (l.4) for further routing process and initialized to *looking*. Then, the query is routed in the network (l.6). The client peer waits for results messages from other peers (l.7). If a *No result* answer is received from a controller peer, the process ends because a part of the user query could not be resolved. Otherwise, the client peer receives the number of potential answering server peers (l.10) and waits for answers (l.11). Documents results are stored (l.13) and global result is returned once every server peer sends its answers (l.17).

Controller Peer The demand keeps the list of potential results (i.e. the results answering resolved vlps). The controller peer updates this list of addresses containing potential results (l.1). For each value-localization-pathwith state set to *looking* having their key value comprised in the key interval of the controller peer (l.2), the addresses are updated by checking segments (l.3) and the value-localization-path state are set to *found*. If a controller peer concludes that no server peer can answer, a *No Result* message is sent to the client peer and the routing process ends (l.5-6). When all segments for this controller peer have been checked, the demand is routed to the controller peer for the next unresolved value-localization-path(l.13). When every value-localization-pathhas been resolved (l10-11) the demand is forwarded to servers peers.

Server Peer A server peer is contacted when its DBF segments have filtered successfully each value-localization-path of the query demand. The server receives the demand for computing results. False positive due to the probabilistic structure of DBF are removed at this phase.

Algorithm 1 Client Send request.

```
Require: XQT: a Query Tree.
Ensure: A set of documents collection
 1: $V \leftarrow$ A set of value-localization-paths
 2: $Wait \leftarrow \oslash$
 3: for value-localization-path $v_i$ in V do
 4:     $D \leftarrow D \wedge$ Order($H_1(s_i)$, Demand($s_i$))
 5: end for
 6: $Route$(D).
 7: if (Receive(W) = No Results) then
 8:     Return $\oslash$
 9: else
10:     $Wait \leftarrow$ NumberOfResults(R)
11:     while ($notAllReceived(Wait)$) $\wedge$ $Receive(W)$) do
12:         if (emptyDoc(W) $\neq \oslash$) then
13:             R $\leftarrow$ R $\wedge$ Document(W)
14:         end if
15:     end while
16: end if
17: return R
```

Algorithm 2 Controller peer.

```
Require: D; the demand.
Ensure: a set of relevant server peer addresses
 1: $R \leftarrow$ Server Peer addresses computed so far.
 2: while P $respOfKey$ $H_1(nextUncheckedVlp)$ do
 3:     $R \leftarrow R \wedge check(nextUncheckedVlp)$
 4:     if ($R = \oslash$) then
 5:         Send "No results" to client peer.
 6:         Terminate process.
 7:     end if
 8: end while
 9: if ($nextUncheckedVlp = \oslash$) then
10:     Route(D, R)
11:     Send R to Client Peer
12: else
13:     Route(D, $H_1(nextUncheckedVlp)$).
14: end if
```

Query Routing We illustrate the routing process in a Chord network of the message at bottom of figure 2 on the figure 3. At beginning, the client peer P1 fills the demand with value-localization-paths and the demand is routed to controller peers. The value-localization-paths are resolved in the order of their key value (computed from H_1 and theme). The demand is routed, using the Chord principles, to con-

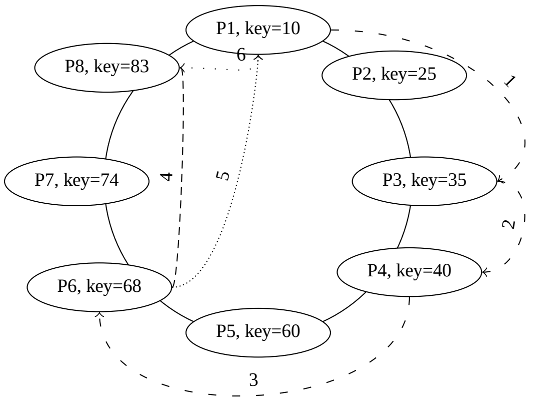


Figure 3: Query Routing.

troller peers P4 (step 1 and 2) and then P6 (step 4) responsible of DBF segment for the two value-localization-paths; server peers addresses successfully filtered are filled in the demand. Next, only server peer P8 filtered by both DBFs (answering the two parts of the query) is contacted (step 4). The client peer P1 waits for incoming results (step 5). Servers peers filtered as potential relevant data sources send results (step 6).

4 EXPERIMENTS

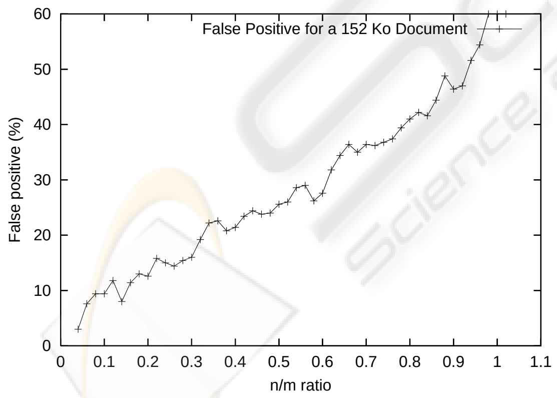


Figure 4: False positive rate.

For the representation efficiency of the DBF model, the first experiment shows the percentage of false positive answers of our DBF. A XML document of 152 Kb is randomly generated, that contains 12260 value-localization-paths. We construct a DBF for this document with 10 hash functions. In Figure 4, each measure is the percentage of false positive for 500 queries when the DBF size varies (the size is determined by the ratio *n/m*). According to the figure, we can deduce that an acceptable percentage of false positive (less than 4%) require a ratio *n/m* equal to 0.04. The compression ratio is 6.79% of the original data.

We compare the connection cost of peers on two P2P network platforms: a Chord network (called *Basic Chord*) indexing XML content as in most KaDop or PathFinder systems and a Chord with DBF (called *DBF Chord*).

Table 1 exhibits in the third column (Messages) the number of messages and the total size of data transiting in the network for indexing peers data (each peer shares data).

In the *DBF Chord*, the number of messages sends for indexing peer data depends on the network size and the number of segments created. In the *Basic Chord*, the number of messages sent depends on the number of keys. As described in our data model, the number of messages remains constant in *DBF Chord* with a growing key number. These results show that our data model requires a bounded number of messages that allows a fast connecting process.

The fourth column (Messages Size) of Table 1 reports the total size (in byte) of data exchanged in the network. In *Basic Chord*, we use messages containing a 4 byte key, and the server peer address. The key size is the average size for value and structure indexing numbering scheme used to address XML elements. The ratio *n/m* for *DBF Chord* is 0.08. We observe that the total size for the *DBF Chord* is extremely small compare to the *Basic Chord* size. In both case, the size is correlated to the number of messages sent in the network.

Finally the fifth and sixth columns show the index size (i.e. stored at controller peers) with respect to the original size. For both, compression ratio is acceptable (20%) and roughly equivalent.

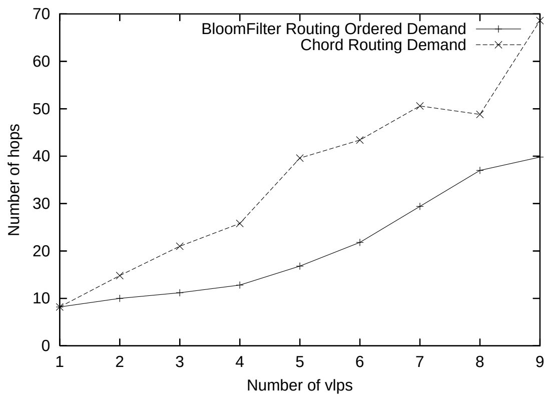


Figure 5: Communication messages for routing a demand.

Table 1: Network Performance : Chord Key Indexing vs. Chord DBF Indexing.

Network		Messages				Data		
Network size (peers)	Number of keys	Number		Size (byte)		Data Size (byte)	Index Size (byte)	
		Chord	DBF	Chord	DBF		Chord	DBF
8	11860	11456	256	400960	36640	147529	34624	28704
16	24480	22728	512	795480	74136	303168	68696	58264
32	47720	41654	1024	1457890	146632	591350	125986	114888

The Figure 5 focus on the number of hops (i.e. number of peers contacted during the process) for resolving a query. Queries are composed of one or more value-localization-paths. The graph compares a *Basic Chord* using the *get(key)* primitive to a DBF network with ordering message. The network using DBF is more efficient in term of hops compared to *Basic Chord*. Indeed, in *Basic Chord* every value-localization-path is sent in the network and a response is systematically returned to the sender. In comparison, a single message is routed in the network using DBF until it reaches the source peers. Then, the source peers send the results to the client peer. These curves show that our new routing protocol requires fewer messages than a traditional approach.

5 CONCLUSION

In this paper, we have proposed a new P2P indexing model based on Bloom Filters. This model is used to index XML document content and structure. We design a Distributed Bloom Filter to summarize peer content. Compared to other proposals, Distributed Bloom Filter is a non dense index, offering a good compression ratio of the original data. We proposed technique to distribute this Bloom Filter on a DHT based network and provide algorithms to localize relevant sources. The main benefice of our approach is to reduce the network communication for connecting, disconnecting and updating peers; few messages are required despite the number of value to index. We also demonstrated that our query routing method reduce the network traffic despite false positive.

Future works are focused on integrating DBF into the XLive mediation architecture (Dang-Ngoc et al., 2005). Peers are mediator connected to the network, publishing data of mediated sources. Data are indexed in the network using Distributed Bloom Filters.

REFERENCES

Abiteboul, S., Manolescu, I., and Preda, N. (2005). Sharing Content in Structured P2P Networks. In *BDA*, pages 51–58.

Bloom, B. H. (1970). Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426.

Dang-Ngoc, T.-T., Jamard, C., and Travers, N. (2005). XLive : An XML Light Integration Virtual Engine. In *BDA*, pages 399–404.

Fan, L., Cao, P., Almeida, J. M., and Broder, A. Z. (2000). Summary Cache: a Scalable Wide-area Web Cache Sharing Protocol. *ACM Trans. Netw.*, 8(3):281–293.

Gardarin, G., Dragan, F., and Yeh, L. (2006). P2P Semantic Mediation of Web Sources. In *ICEIS (1)*, pages 7–15.

Halevy, A. Y., Ives, Z. G., Mork, P., and Tatarinov, I. (2003). Piazza: data management infrastructure for semantic web applications. In *WWW*, pages 556–567.

Jagadish, H. V., Ooi, B. C., and Vu, Q. H. (2005). BATON: A Balanced Tree Structure for Peer-to-Peer Networks. In *VLDB*, pages 661–672.

Koloniari, G., Petrakis, Y., and Pitoura, E. (2003). Content-Based Overlay Networks for XML Peers Based on Multi-level Bloom Filters. In *DBISP2P*, pages 232–247.

Ratnasamy, S., Francis, P., Handley, M., Karp, R. M., and Shenker, S. (2001). A Scalable Content-addressable Network. In *SIGCOMM*, pages 161–172.

Rousset, M.-C., Adjiman, P., Chatalic, P., Goasdoué, F., and Simon, L. (2006). Somewhere in the semantic web. In *SOFSEM*, pages 84–99.

Rowstron, A. and Druschel, P. (2001). Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218:329–350.

Stoica, I., Morris, R., Karger, D., Kaashoek, F., and Balakrishnan, H. (2001). Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160.