Bloom Filter-Based Keyword Search over XML Data in Structured Peer-to-Peer Systems

Weimin He
Department of Computing and New Media Technologies
University of Wisconsin-Stevens Point
Stevens Point, WI, USA
Email: whe@uwsp.edu

Teng Lv
Teaching and Research Section of Computer Artillery Academy
Hefei, Anhui, China
Email: lt0410@163.com

Abstract—With the popularity of Extensible Markup Language (XML) as the new standard of data exchange on the web, a large number of data sources are represented or encoded in XML format. Therefore, querying and searching XML data on the web has attracted much attention in the database literature. The new emerging Peer-to-Peer (P2P) computing model has fueled the autonomous data sharing over the Internet in a more flexible fashion. Needless to say, XML data retrieval in P2P systems has become attractive to professionals in both research and industrial communities. In this paper, we propose a Bloom-Filter based keyword search framework for XML data retrieval in structured P2P systems. We designed an efficient Bloom-Filter based XML indexing scheme for XML data retrieval in structured P2P systems. We also developed an effective keyword search algorithm over our Bloom-Filter encoded XML indexes. The experimental results demonstrate that our novel indexing scheme is much more efficient compared to the traditional full-indexing scheme, our keyword search algorithm is efficient in terms of query response time, and our system is scalable in terms of data size and network size.

Keywords—Database, XML, Peer-to-Peer.

I. INTRODUCTION

Over the past one decade, due to its simplicity and flexibility, Extensible Markup Language (XML) is rapidly gaining in popularity as a universal data format for data exchange and integration on the web. A volume number of web data sources are represented as XML documents or fragments on the Internet. A large number of web services are also encoded in XML format. XML has undoubtedly become the new standard for data representation and data exchange on the web. Therefore, querying and searching XML data on the web has attracted much attention in the database literature.

Meanwhile, peer-to-peer(P2P) systems have emerged as a new paradigm for large scale data sharing over the Internet. A typical P2P system consists of a large number of nodes, called peers, that share data and resources from each other 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. Compared with client-server systems, P2P systems are more scalable, fault-tolerant and easier to deploy. Since no central coordination exists in a P2P system, there is no central point of failure. In addition, network resources can be fully utilized and shared, and the server workload can be distributed among all the peers in the system. Due to the above advantages, P2P systems have made inroads for various content distribution and service discovery applications, such as music file sharing, live P2P internet TV, and instant messaging.

In consequence, searching and querying distributed XML data in P2P systems have attracted significant interests in the DB community. However, the keyword-based search over distributed XML data in structured P2P systems poses more challenges than the keyword-based search over plain text documents in the traditional client-server models. That is because XML data are hierarchical trees instead of plain text documents and the evaluation of keyword queries over these data are more complex and if not optimized, may involve routing and processing massive data in P2P systems. Under distributed P2P settings, the key issue to be addressed is how to design efficient meta-data indexing scheme to summarize the original XML data, develop effective keyword search algorithm to evaluate keyword queries over those meta-data, and finally return the list of XML document hits as query answers to the user.

Many efforts have been made to propose new meta-data indexing schemes and efficient query evaluation algorithms to query distributed XML data in P2P networks [5], [7], [9], [14]. Some work focuses on indexing and querying XML data in unstructured P2P networks. In [9], the authors proposed two multi-level Bloom filters, termed Breath Bloom Filter and Depth Bloom Filter, to summarize the structure of an XML document for efficient path query routing in unstructured P2P networks. Kd-synopsis [14] is another routing synopsis based on length-constrained FBSimulation relationship, which allows the balancing of the precision and size of the synopsis according to different space constraints on peers with heterogeneous capacity. Other researchers are interested in query processing of XML data in structured P2P networks. In [5], the authors proposed XP2P, which employs concrete paths in an XML document to index XML fragments. XP2P can answer simple linear XPath queries efficiently, but does not support complex XPath queries with conditions or search predicates. In [7] Galanis et al. proposed an meta-data indexing scheme and query evaluation strategies for distributed XML data in
structured P2P networks. In their framework, a distributed catalog service is distributed across the data sources themselves and is responsible for routing user queries to the relevant peers. In [10], Li et al. presented a scheme for searching XML documents by keywords in DHT-based structured P2P networks. They constructed an inverted index for XML documents in an DHT-based P2P network. To achieve that goal, they proposed to decompose XML documents and store the resulting fragments in the DHT network. Given a query in terms of keywords, their framework can find respective XML fragments that subsume given keywords. They addressed this problem by finding SLCAs (Smallest Lowest Common Ancestors) for the keywords with the help of the DHT-based inverted index.

In this paper, we present a novel framework for indexing and searching distributed XML data in DHT-based structured P2P networks. First, we propose an efficient Bloom Filter-based meta-data indexing scheme. Then, we develop an effective keyword search algorithm over our indexed meta-data to locate XML document hits which can satisfy the user query in the DHT network. Finally, we experimentally validate the efficiency of our indexing scheme, the effectiveness of our keyword search algorithm, and the scalability of our system.

II. PRELIMINARIES

A. DHT-based Structured P2P Networks

In a DHT-based structured P2P system, the location of data is determined by some global scheme. More specifically, as a data item $I$ is published by some node, a global hash function is used to map the key $K$ of $I$ to an IP address of a node $N$ in the P2P network and $I$ is placed on node $N$. One popular mapping approach is called consistent hashing [13], in which both a key and node Id are mapped to the same identifier space. Identifiers are ordered in an identifier circle modulo $2^m$, where $m$ is the length of an identifier. The data item $I$ associated with the key $K$ is assigned to its successor node, which is the first node whose identifier is equal to or follows the identifier of $K$ in the identifier space. In our framework, we employ Pastry [11] as our P2P back end, in which the key $K$ is assigned to the node whose Id is closest to the Id of the key. The Id of $K$ is derived from a base hash function such as SHA-1 [13].

B. Bloom Filters

A Bloom Filter [3] is a compact probabilistic data structure which is used to test whether an element might be a member of a set. A Bloom Filter is implemented using a bit vector with $m$ bits. For an empty set, all bits of the Bloom Filter are set to zero. If the set $E$ is non-empty, $k$ bits are set to one for each element $e$ in the set $E$. The indices of these $k$ bits are obtained using $k$ hash functions $H_k: E \rightarrow \{0, \ldots, m\}$. Bits can be set to one multiple times. In order to test whether an element $e$ may be a member of the set, the $k$ bits $H(e)$ are checked. If all $k$ bits are set to one, $e$ is considered a potential member of the set and the Bloom Filter test returns true, otherwise false [4].

III. BLOOM FILTER-BASED DATA INDEXING

A. Extraction of Structural Summary

In order to facilitate the publishing and indexing of XML data in the DHT network, we first extract a data synopsis termed Structural Summary (SS), which is a structural markup that captures all the unique paths in an XML document. Figure 1 gives an example XML document that represents the excerpted book information from Amazon. The structural summary of the example XML document is shown in Figure 2. Each node in an SS has a tagname and a unique Id. Note that one SS node may correspond to more than one elements in the original XML document. For example, the node book in Figure 2 corresponds to three book elements in Figure 1. As an XML document is published by a peer, the SS is extracted from the document, which serves as the basis for meta-data indexing in the DHT network. We exploit Java SAX API to extract the structural summary from an XML document. Due to the space limitation, the structural summary extraction algorithm is omitted here.

B. Data Publishing and Indexing Based on Bloom Filters

Since an XML document may contain thousands of elements with the same names and each element may contain hundreds of terms, fully indexing XML data in the DHT
network is not feasible in terms of both data publishing and query response time. In addition, due to the dynamic characteristics of P2P networks, full-indexing scheme may also lead to the overload of data update. In order to address the above issues, we propose a Bloom Filter-based meta data indexing scheme.

More specifically, as an XML document $D$ is published, for each leaf node $e$ in the structural summary of $D$, we construct a triple $<bf; peerId; docId>$ to summarize the textual data contained in the element $e$ in the document $D$. In the triple above, bf is a Bloom Filter that summarizes the textual data enclosed by $e$. Each term enclosed by $e$ in $D$ is hashed to a bucket in bf and the corresponding bit is set to 1. And docId is the document ID of $D$ and peerId is the peer ID of the peer that owns the document $D$. We use tag, which is the tag name of $e$, as the DHT key to route the triple to the peer $P_{tag}$, who is responsible for storing the data related to the key tag. In the local index of $P_{tag}$, we use tag as the key to store the triple $<bf; peerId; docId>$.

IV. KEYWORD QUERY EVALUATION

A. Specification of Keyword Search Queries

Since XML data is hierarchical and recursive semistructured data instead of flat and plain textual data, we believe that it would be better to consider the structure features when designing the keyword search queries. We extend the traditional simple keyword search query syntax with a construct named tag:term pair, where tag is an element in an XML document and term is a text term that may appear in the element tag. In another word, A keyword in our framework is a tag:term pair instead of a single term. In consequence, our keyword query takes the following form: 

$tage : \text{term AND tag} : \text{term AND} :: : \text{AND tag} : \text{term}$. As a running example used throughout the paper, the following keyword search query, $Q$:

\[Q = \text{title}:\text{XML AND author}:\text{David AND year}:2003\]

searches for XML documents that have a title element containing the term “XML”, an author element containing the term “David” and a year element containing the term “2003”.

#### Algorithm 1: Keyword Query Evaluation in the DHT Network

**Input:** 
Q /* User query that contains a list of tag-term pairs */
id /* Pastry node ID of query client */
docList /* Current list of document hits */

**Output:** 
resultDocList /* the list of document hits that satisfy the keyword query */

```
if( currentKeywordIndex < Q.size )
tag := Q.get( currentKeywordIndex ).tag;
term := Q.get( currentKeywordIndex ).term;
localList := bfSynopsis list returned by accessing the local index using tag as the key;
for i := 1 to localList.size do
   synopsis := localList.get(i);
   Hash term to a bit b in the Bloom Filter bf in synopsis
   if( b=1 )
      hit := < synopsis.peerId, synopsis.docId >;
      if( currentKeywordIndex = 0 )
         docList.add(hit);
      else
         found := false;
         for i := 0 to docList.size do
            h := docList.get(i);
            if( h.peerId = hit.peerId AND h.docId = hit.docId )
               found := true;
               break;
         end if;
         end for;

      if( found = false )
         docList.remove(hit);
         end if;
      end if;
      end if;
      end for;
      if( currentKeywordIndex + 1 = Q.size )
         resultDocList := docList;
         Route resultDocList to the query client whose Pastry NodeID is id;
      else
         Route docList to the next peer for further processing;
      end if;
   end if;
end if;
```

Note that the goal of the proposed framework in this paper is to locate relevant XML data sources in structured P2P networks. Thus the query answers of a keyword search query in our framework is a list of document hits that may satisfy the query instead of the actual XML fragments.

B. Keyword Query Evaluation Algorithm

In order to evaluate a keyword user query, we have developed an efficient keyword query evaluation algorithm that can route the user query among relevant peers, collect qualified document hits along the way, and finally return the resulting document hits to the user. Our keyword query evaluation algorithm is shown in Algorithm 1. We illustrate Algorithm 1 using our running example query $Q$. Figure 3 demonstrates how the keyword query $Q$ is evaluated in the DHT network. First, the user poses the query $Q$ on the Querying Peer. Then the pair (title:XML) and the initial empty resultDocList are routed to the peer $P$(title). On the peer $P$(title), “title” is used as the key to access the local index to obtain a list of triples (bf, peerId, documentId). “XML” is then mapped to a bit in bf and the pair (peerId, documentId) will be added in the resultDocList if the bit is set to one. After that, the intermediate resultDocList and the pair (author:David) are routed to the peer $P$(author). On the peer $P$(author), “author” is used as the key to access the local index to obtain a list of triples (bf, peerId, documentId). “David” is then mapped to a bit in bf and the pair (peerId, documentId) will be added in the resultDocList if the bit is set to one. After that, the updated intermediate resultDocList and the pair (year:2003) is routed to the peer $P$(year). On the peer $P$(year), “year” is used as the key to access the local index to obtain a list of triples (bf, peerId, documentId). “2003” is then mapped to
TABLE I. EXAMPLE XMARK QUERIES

<table>
<thead>
<tr>
<th>Query</th>
<th>Query Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>payment:creditcard</td>
</tr>
<tr>
<td>Q2</td>
<td>location:United</td>
</tr>
<tr>
<td>Q3</td>
<td>payment:creditcard location:United</td>
</tr>
<tr>
<td>Q4</td>
<td>payment:cash location:United</td>
</tr>
<tr>
<td>Q5</td>
<td>payment:cash payment:United</td>
</tr>
<tr>
<td>Q6</td>
<td>text:earth type:regular</td>
</tr>
<tr>
<td>Q7</td>
<td>payment:cash location:United text:earth</td>
</tr>
<tr>
<td>Q8</td>
<td>payment:check location:States text:gold</td>
</tr>
<tr>
<td>Q9</td>
<td>payment:creditcard payment:cash location:States</td>
</tr>
<tr>
<td>Q10</td>
<td>text:heat type:regular text:gold</td>
</tr>
</tbody>
</table>

A bit in bf and the pair (peerId, documentId) will be added in the resultDocList if the bit is set to one. Finally, the resultDocList is routed back to the Querying Peer and the query answer is returned to the user.

V. EXPERIMENTAL EVALUATION

A. Experimental Setup

We have built a prototype system based on our framework using Java (J2SE 6.0). Our DHT-based P2P back end is FreePastry 2.0 [11], which is responsible for routing various messages associated with data publishing and query evaluation to specific peers. On each peer, Berkeley DB Java Edition 3.2.13 [2] is employed as the local indexing DB. Our experiments were carried out on a laptop running WindowsXP with 2.26GHz CPU and 3G memory. The maximum heap size for Java Virtual Machine was set to 512MB. We conducted several sets of experiments to measure the efficiency of our meta-data indexing scheme, the effectiveness of our keyword search algorithm, and the scalability of our framework. Our experimental data are synthetically generated from XMark benchmark [15], from which data synopses are extracted and populated over the DHT network. For all experiments except those measuring system scalability, the size of the data set is 55.8MB and the number of documents is 11500. In our experiments we create 1150 Pastry nodes and let each node publishes 10 XML documents over DHT network. Since scalability experiments require large data volumes, we multiply the above data collection by copying when larger volumes are needed. Each peer owns its database to index the meta-data. We designed 10 queries over XMark data to conduct query-related experiments and the queries are shown in Table I. Note that the logical “AND” operator is omitted in each query in Table I.

B. Efficiency of Bloom Filter-Based Indexing Scheme

In this experiment, we measure the efficiency of our Bloom Filter-based meta-data indexing scheme (BFI). More specifically, we compared the efficiency of our indexing scheme BFI with that of traditional full indexing scheme (FI). We first compared the data publishing time between BFI and FI. The result is shown in Figure 4, from which we can see that the average peer publishing time of BFI is only about 35% of that of FI. We then compared the index size between BFI and FI. The result is shown in Figure 5, from which we can see that the average peer index size of BFI is only about 36% of that of FI. Finally, we compared the query evaluation time between two schemes for all the queries in Table I. From Figure 6, we can see that for all the queries, BFI is much faster than FI in terms of query response time. The above experiments demonstrate our Bloom Filter-based indexing scheme is efficient in terms of data publishing time, index size and query response time.

C. Scalability Measurement

In this experiment, we measure the scalability of our system. We first varied the data size from 56MB to 560MB to measure the total data publishing time. As we can see from Figure 7, as the data size increases, the data publishing time scales almost linearly in terms of the size of published...
data. We then varied the number of peers in our system to measure the total data publishing time. As we can see from Figure 8, as the network size increases, the total data publishing time increases smoothly with the increasing number of peers. The above results indicate that our system scales gracefully in terms of both data size and network size.

VI. CONCLUSION AND FUTURE WORK

We have presented a novel framework for answering keyword queries in DHT-based P2P networks. We first designed efficient Bloom Filter-based meta-data indexing scheme to summarize original XML data and populate these meta-data in a DHT-based P2P system. Then we developed an effective keyword search algorithm over the distributed XML data in the DHT network. Finally, we conducted experiments to demonstrate the efficiency of our indexing scheme and keyword search algorithm in terms of data publishing time, index size, and query response time. As our future work, we plan to adapt the classical $tf \times idf$ scoring in IR to develop an effective ranking function for our framework. The development of ranking function can be challenging and interesting due to the distributed and dynamic features of P2P systems.

This work is supported by the Faculty & Program Development Grant from the University of Wisconsin-Stevens Point, USA, 2010.

ACKNOWLEDGMENT

This work is also supported by the Natural Science Foundation of Anhui Province, China(No. 070412057).

REFERENCES