Cooperating Peers for Content-Oriented XML-Retrieval

Judith Winter
Institute of Computer Science / Telematics
J.W.Goethe-University Frankfurt, Germany
winter@tm.informatik.uni-frankfurt.de

Oswald Drobnik
Institute of Computer Science / Telematics
J.W.Goethe-University Frankfurt, Germany
drobnik@tm.informatik.uni-frankfurt.de

Abstract

Semi-structured documents formatted with the extensible markup language (XML) are gaining wide use by a whole range of applications including E-Commerce, E-Business, E-Science, Digital Libraries (DL), File Sharing, and in the last years especially by applications for Peer-to-Peer (P2P) systems. P2P architectures have been identified as an efficient means of ad-hoc collaboration and information sharing among large, diverse, and dynamic sets of user. However, current P2P search engines for XML-documents lack the use of information retrieval methods to efficiently search XML collections for relevant information.

This article proposes a search engine for P2P systems that applies an extension of the vector space model and exploits structural information to compute relevance of XML-documents, and thus may significantly improve retrieval performance. We concentrate on the cooperation of peers that perform a distributed query execution through cooperated retrieval and ranking of dynamic XML documents. The interaction between the participating peers is based on a structured P2P-network and uses an adaption of the DHT-algorithm Kademlia.

1. Introduction

Peer-to-Peer (P2P) systems are a form of distributed computing that involves a possibly large set of autonomous computing nodes – the *peers*. These cooperate to share resources such as computing power or storage. Unlike traditional client/server systems, peers can decide autonomously which services and resources to contribute to the system and which ones to make use of, hence acting as both a client and a server. P2P systems are not centrally controlled, but self-organizing. Therefore, they bear the potential to realize robust and fault-tolerant systems that may scale theoretically to unlimited numbers of participating nodes [12]. A P2P-network can be organized as a *structured overlay network,* in which the set of cooperating peers act on a distributed data structure with well-defined operations, e.g. a *Distributed hash table (DHT)* supporting joining, leaving, and routing between the peers [4].

Schema-based P2P-networks [12] provide techniques for the lookup of semi-structured documents such as those modelled with the *extensible markup language (XML)* [15]. These networks allow for exact or partial matches and take into account hints about the desired document structure. So far, however, they do not provide means to compute the relevance of documents and thus should adapt methods from *information retrieval* (IR) such as the *vector space model* [5].

IR in P2P-networks concerning *unstructured* documents is an emerging field of research and deals with locating distributed relevant documents. A P2P architecture for information retrieval is proposed in [1]. A survey about looking up data in P2P systems presents search techniques using DHTs in [3]. Algorithms for IR in P2P benefit from database systems performing distributed execution of exact queries; classic IR systems ranking retrieved results by relevance; and distributed systems where DHTs support efficient lookup of objects [10]. An example for a decentralized non-flooding P2P IR system is described in [13]. Additionally to dealing with unstructured documents, there are approaches for multimedia retrieval in P2P systems. However, no solutions for XML IR in P2P exist so far.

Content-based XML-Retrieval (or XML Information Retrieval), i.e. applying information retrieval methods to the retrieval of XML-documents, takes advantage of the self-describing structure of XML and can substantially improve the retrieval performance. For example, *content and structure (CAS) queries* can enable users to specify what structure the requested relevant content can have and *retrieval units* can consist of entire documents or only the most relevant parts of a relevant document. The aim is to find the smallest retrieval unit that is highly relevant in terms of *specificity*, i.e. the extent to which a retrieval unit focuses on the intended topic [6]. The challenges introduced by XML IR include extracting and indexing structural data; ranking can incorporate both content relevance and *structural similarity*, which is the resemblance between the structure given in the query and the structure of the document. The INitiative for the Evaluation of XML-Retrieval (INEX) provides a platform for evaluating algorithms for content-oriented XML-Retrieval [6]. However, these algorithms are currently all based on the classic client/server architecture. An overview of existing approaches can be found in [2].

Our proposal for a P2P search engine for XML-documents is based on an architecture developed for content-oriented XML-Retrieval in P2P [14]. To the best of our knowledge, no other approaches yet exist.

2. Peer-based XML-Retrieval

In this section, we propose a search engine that is based on cooperating peers using information retrieval techniques to rank XML-documents. We therefore outline the aims of the search engine, the peer architecture and the interaction between its distributed components.

2.1. Retrieval goals

Our focus is on efficient retrieval at querying time, and therefore we accept the indexing to be possibly quite exhaustive. The proposed search engine aims for document providers who stay online for quite a while whereas free-riders who do not contribute documents might participate by offering resources such as space for the distributed indexes.

The search engine to be built will enable CAS queries, i.e. users will be able to give structural hints about the desired content. Structural information is therefore regarded in the ranking model and thus included into the indexing strategy as well. Retrieval results do not always match a given query exactly, but information retrieval techniques are applied to compute relevance of retrieval units. These can be entire documents or dynamic documents, i.e. any relevant sub-tree of a relevant document. The majority of queries are expected to consist of more than one single query term; this is considered in the indexing and retrieval process. Common P2P-properties such as scalability, robustness, and load balancing are achieved by self-organisation mechanisms. Special care is taken to minimize bandwidth consumption and to enable parallel computing. The index is self-reorganizing, too.

A major challenge is to distribute all information in an efficient way such that participating peers can easily access it while cooperating to perform a distributed execution of a given query.

2.2. Peer architecture

The proposed architecture of each peer is shown in figure 1. It is founded on a general concept for XML search engines in P2P-networks and has been derived from a component-based architecture for such search engines [14]. The numbered arrows in figure 1 denote the interaction and data flow between different components of cooperating peers managed by the P2P-layer. Details about interaction and data flow are explained in section 2.3-2.4 and shown in figure 2.

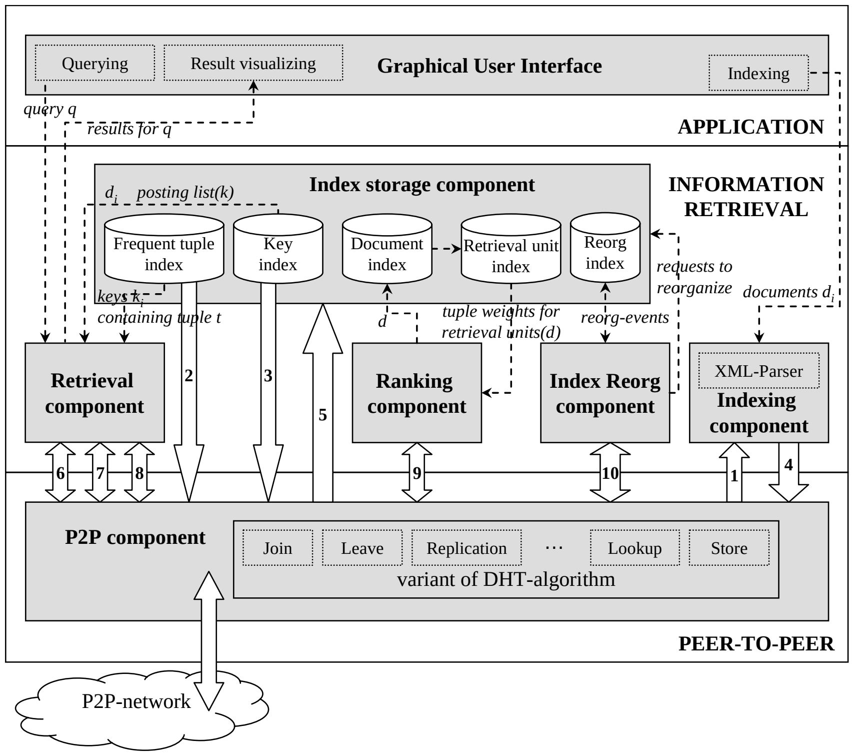


Figure 1. Architecture of a peer

A user accesses the application via a graphical user interface (GUI) which is part of the APPLICATION-LAYER and manages all interaction between user and INFORMATION RETRIEVAL LAYER. Interaction includes indexing of XML-documents, querying, and visualizing of results. All components of a peer cooperate with components of other peers to fulfil these tasks. Lookup and storage of necessary information as well as the exchange of requests between cooperating peers are handled by the PEER-TO-PEER LAYER.

2.3. Information Retrieval layer

The Information Retrieval layer is composed of components for indexing, retrieval, ranking, index storage handling, and reorganisation. These components interact as follows.

The INDEXING COMPONENT performs the indexing process. The XML-documents to be indexed are parsed such that all contents (in respect of terms) together with their structure are extracted. We denote a term's *structure* as the path from the document root element to the term element in the XML-document tree, and express it with XPath [16]. Depending on their frequency in the global collection, extracted term-structure tuples are merged into sets of tuples. Each set represents an index *key*: a key can either consist of a single *rare* tuple, in which case the tuple's frequency in the document collection does not exceed a frequency threshold. Or, the key consists of a set of *frequent* tuples if each tuple is frequent but the combination of tuples is rare. Key frequencies used in the indexing process are requested via the P2P-layer and received from the global key index. While extracting the keys, term statistics are collected for the ranking process. These statistics are not limited to static document statistics but include information about dynamic sub-documents, too; for each document, term statistics for several of its potential retrieval units are computed. The following information is then distributed via P2P-layer to corresponding peers for insertion into different indexes:

- ♦ keys together with their posting lists that contain documents in which the keys appear (→ key index),
- ♦ updated key frequencies (→ key index),
- ♦ document IDs with links to the documents' retrieval units (→ document index),
- ♦ term statistics for the retrieval units of each document (→ retrieval unit index),
- ♦ frequent terms and their structure with links to keys that contain these tuples (→ frequent tuple index).

The indexed documents can be distributed over the P2P-network as well. However, we propose the option of having full control over the accessibility of provided documents to guarantee the autonomy of the document provider. This means to store documents locally at the provider's peer and to avoid replicas that could be found in the network long after the provider has chosen to explicitly delete those documents. Each provider then has both full control and responsibility for the availability of documents.

The RETRIEVAL COMPONENT manages query distribution and result retrieval. A given query q is split into existing keys, about which information is requested from the global key index, and the frequent tuple index. For each identified key, the retrieval component sends a request to retrieval components of other peers. These peers will use their local keys' posting lists to redirect the requests to ranking peers holding term statistics for retrieval units of documents containing those keys. All retrieval units computed as relevant are collected by the querying retrieval component and evaluated. The ranked results are sent to the GUI for visualization.

The RANKING COMPONENT performs the relevance computing. For a given query q, relevance is computed in parallel on all peers holding documents potentially relevant for the query. A peer p's ranking component can locally access the term statistics of all retrieval units of the documents assigned to p. Hence, for a query redirected to peer p, the ranking component can compute the relevance of these retrieval units. The computation is based on the vector space model, extended by applying structural information of XML elements. It is performed only for documents that contain at least one key of the query.

INDEX STORAGE COMPONENT: This component allocates information stored in the different local indexes of a peer and passes it on to requesting local or distributed components.

INDEX REORG COMPONENT: In addition to implementing self-reorganization mechanisms for the P2P-network, we propose similar mechanisms for index reorganization. This task is managed by the index reorg component with information from the reorg index. Reorg-events to be dealt with include, for example, missing or deleted documents that must be eliminated from key index, document index and retrieval unit index; keys that have become frequent must be recombined to rare sets of term-structure tuples and updated in the key index. These events are notified to the different index reorg components of peers assigned to keys, documents, peers etc. causing the events. Reorganization of specific local indexes will be performed periodically or when thresholds are exceeded.

2.4. Peer-to-Peer layer

The Peer-to-Peer layer handles all tasks necessary for the cooperative execution of a query as follows.

Information such as keys, posting lists, document IDs, retrieval units, term statistics and frequencies are distributed over the P2P-network and *stored* on their assigned peers. For the assignment of keys to peers, the hash values of a key's terms are used without hashing their structure. This guarantees that similar keys with identical terms but different structures are stored on the same peer. As a consequence, redundant redirected requests can be reduced or avoided by reducing the number of peers sending requests for identical documents; additionally, the retrieval component can locate similar keys easily on the same peer.

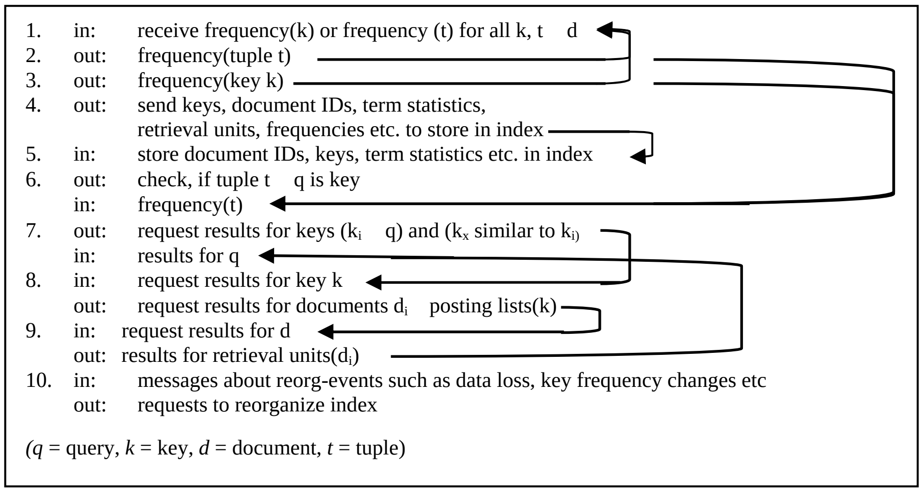
Due to an efficient organization of the P2P-network with a DHT algorithm, e.g. derived from the Kademlia-protocol [7], *lookup* of the stored information can be performed in O(log(N)), with N being the number of participating nodes.

If a peer *joins* the network, the existing indexes will be redistributed to include the new peer so as to achieve load balancing. If a peer *leaves* the network, all its data will be redistributed among the remaining peers.

If a peer leaves unintentionally, e.g. due to network problems, its data will be *recovered* and redistributed using replicas stored on other peers. Different replication strategies can be applied for the different indexes. For instance, losing posting list information of keys is critical as this implies the loss of connections between terms and documents. In contrast, losing weights of retrieval units only reduces the specificity of the results (they may become too big or too small) as long as some retrieval units of each document still are accessible. Thus, the key index should be secured by a higher replication factor than the retrieval unit index.

2.5. Interaction and data flow

The interaction between IR components of cooperating peers managed and transferred by the P2P layer as explained in the previous sections is shown in figure 2. The numbers refer to the arrows in figure 1.



1. in: receive frequency(k) or frequency (t) for all k, t d
2. out: frequency(tuple t)
3. out: frequency(key k)
4. out: send keys, document IDs, term statistics, retrieval units, frequencies etc. to store in index
5. in: store document IDs, keys, term statistics etc. in index
6. out: check, if tuple t q is key
   in: frequency(t)
7. out: request results for keys (k_i q) and (k_x similar to k_i)
   in: results for q
8. in: request results for key k
   out: request results for documents d_i posting lists(k)
9. in: request results for d
   out: results for retrieval units(d_i)
10. in: messages about reorg-events such as data loss, key frequency changes etc
    out: requests to reorganize index

(*q* = query, *k* = key, *d* = document, *t* = tuple)

Figure 2. Interaction and data flow details

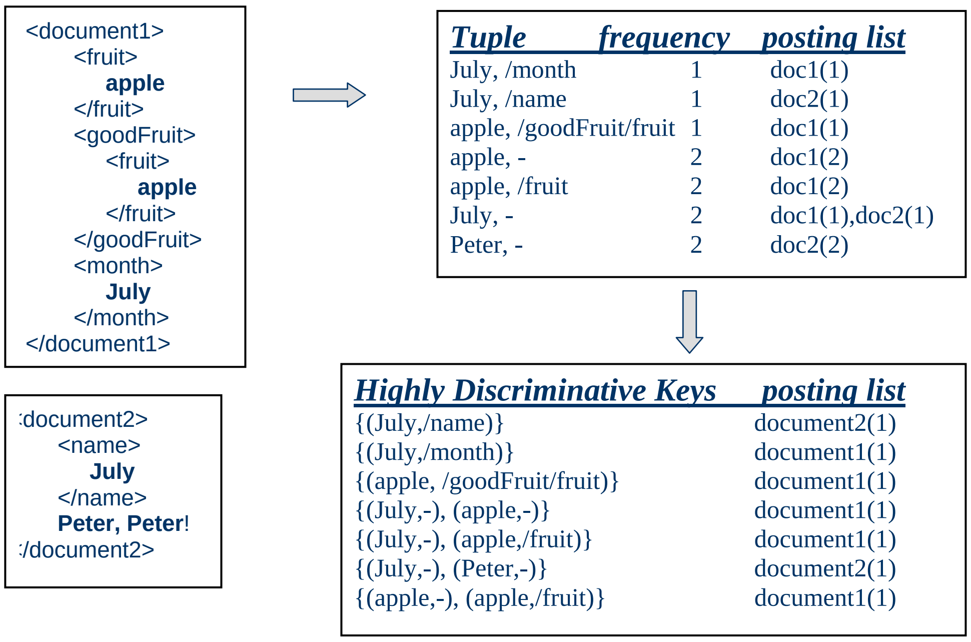
3. Cooperative retrieval process

In this section, the distributed retrieval by cooperating peers is illustrated. We outline the indexing and ranking model on which the retrieval is based.

Large communication overhead and high bandwidth consumption are main problems for information retrieval in a distributed environment. Expensive joins of long posting lists can consume much computation time and bandwidth. We thus aim at reducing the number and size of messages sent between peers as well as the number of calculations performed at querying time. In particular, we consider that at least 85% of all queries are multi term queries [7], and therefore use pre-joint term-combinations for indexing and retrieval.

3.1. Indexing and ranking model

Instead of single term indexing, we apply a key-based indexing strategy that has been derived from [9] and extended by structural information [14]. For each index term, a tuple (term, structure) is stored in the index for each of the term's structures. If the tuple is rare, it can be used as a key and will be stored in the key index together with its posting list. If the tuple is frequent, we regard it as too unspecific and require it only be used in multi term queries, i.e. specified by further query terms. Frequent tuples are recursively combined with other frequent tuples until the new tuple combination is rare, hence specific, and can be stored as a key. The combination of tuples is limited to those occurring close to each other in one document in order to achieve coherence between the tuples of the same key. For each frequent tuple, a link to each key that contains this tuple is created in the frequent tuple index; correspondingly, it is easy to find a key covering a given tuple. The frequency of a key is limited by a threshold PL_{max}; posting lists will hence be limited to this threshold, too. We obtain a key index that is much longer than a single term index but has much shorter entries with a maximum size of PL_{max}.



```
<document1>
     <fruit>
          apple
     </fruit>
     <goodFruit>
          <fruit>
               apple
          </fruit>
     </goodFruit>
     <month>
          July
     </month>
</document1>
```

```
:document2>
     <name>
          July
     </name>
     Peter, Peter!
:/document2>
```

Tuple	***frequency***	***posting list***
July, /month	1	doc1(1)
July, /name	1	doc2(1)
apple, /goodFruit/fruit	1	doc1(1)
apple, -	2	doc1(2)
apple, /fruit	2	doc1(2)
July, -	2	doc1(1),doc2(1)
Peter, -	2	doc2(2)

Highly Discriminative Keys	***posting list***
{(July,/name)}	document2(1)
{(July,/month)}	document1(1)
{(apple, /goodFruit/fruit)}	document1(1)
{(July,-), (apple,-)}	document1(1)
{(July,-), (apple,/fruit)}	document1(1)
{(July,-), (Peter,-)}	document2(1)
{(apple,-), (apple,/fruit)}	document1(1)

Example 1. Extracting keys from documents

Example 1 shows the construction of the key index for two documents. Terms are indexed with all their structures including indexing without tags (-). The latter case is important, as a user is not always able or willing to give structural hints in a query. However, terms indexed without structure have an advanced probability of being frequent. In example 1, threshold PL_{max} was set to 1 (which is of course unrealistically low), so that no posting list will be greater than 1 and all tuples appearing more than once will be regarded as frequent. This is true for (July,-), (Peter,-), (apple, /fruit), and (apple,-). Those frequent tuples that appear close to each other in the same document are combined such that they become rare (frequency of 1).

We apply a ranking model that is an extension of the vector space model [14]. The relevance computing takes into account structure similarity between the structure expressed in a given query and the structure of a relevant document. Retrieval units are dynamic documents, i.e. they can be a relevant document or any relevant sub-tree of a document. For the calculations, we use statistics such as the term-structure weights of retrieval units for all documents.

3.2. Distributed retrieval

Indexing and ranking as outlined above use distributed data but could also be performed on a client/server architecture. We now describe, how retrieval components in a P2P-network can cooperate to perform retrieval and ranking in parallel by sharing storage and computation power.

The retrieval component of a querying peer p_q manages the distribution of relevance ranking to cooperating peers, collects and evaluates their results, and prepares the results for presentation in the GUI.

For a given query q with query tuples (term, structure), peer p_q first checks with the key index, if the query tuples in total form an existing key. This would be the optimal case, as the posting lists for the query would have already been computed. The query therefore would be routed to the peer p_k assigned to this key and located by hashing the key's terms. Peer p_k has one posting list for each structure of the key terms. The posting lists of all structures with similarity to the given query structures would be merged, using structure similarity functions and mechanisms as proposed in [14]. The query would then be redirected to each peer p_d responsible for at least one document of the merged posting list, together with parts of the posting list (at least one document entry per peer). As each document is assigned to exactly one peer p_d, messages are sent to at most PL_{max} peers for each key k_i and each key k_{i_x} similar to k_i. The number of these messages can be significantly reduced by summarizing requests regarding identical documents and target peers. Peers p_d will be located in the P2P network by the hash value of the document ID, which is the unique document location, e.g. \\host\path\documentname). Each peer p_d will locally compute the relevance of all retrieval units that belong to the received part of the posting list, using statistics stored in the local retrieval unit index. IDs of all retrieval units with high relevance will then be sent back to the retrieval component of the querying peer p_q together with additional information such as information about the containing document, first words of the retrieval units, relevance value etc. Peer p_q will collect the results of all ranking peers p_d and merge them. A list of the topmost results, ordered by relevance, will appear in the GUI.

What if the given query tuples do not form an existing key? In this case, the query would be split recursively into tuple sets that are existing keys, starting with sets of size |query terms|-1 as key candidates. For each located key, the procedure described above would be executed. For the worst case of key size=1, we expect 3.5 requests for keys per query, as this is the average query size [7].

Figure 3 illustrates the distributed execution of a query. Dashed arrows denote data flow between components; solid arrows denote the locating of peers that are assigned to the item (key or document) attached to the arrow. Posting lists are denoted by *pl*. Query q can be split into keys k_1, k_2, and k_3. For each key k_i, a request is sent to the retrieval components assigned to k_i, which in turn sends requests for each document of the posting list of each received key plus each key that is *similar* to a received key. In figure 3, k_{3_1} and k_{3_2} are assumed to be similar to k_3. The ranking component of peer p_{d_4} receives a request for document d_4 from k_{3_2}'s posting list and returns relevant retrieval units for d_4 to the querying peer p_q.

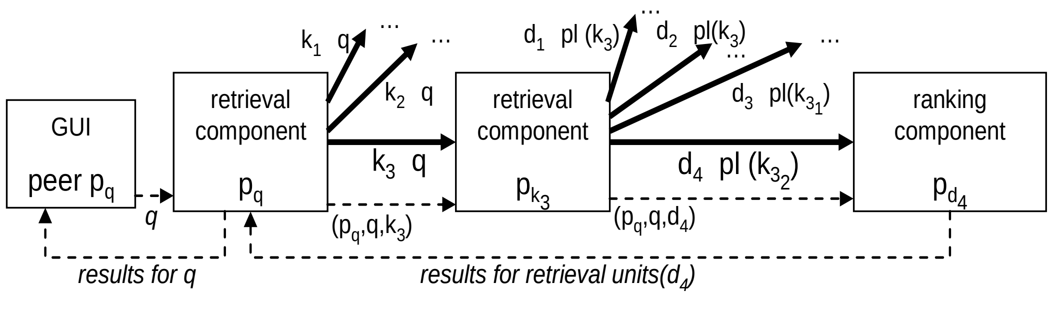


Figure 3. Distributed query execution

If set size = 1 is reached in the process of splitting q into keys, and a query tuple t still cannot be assigned to a key, a list of all keys that cover this tuple will be retrieved from the frequent tuple index and sent to the GUI. The user can then choose which one of the keys specifies tuple t best, or decide to drop it due to its non-specificity. There is the option of retrieving *all* documents containing t, as a common search engine based on single term indexing would do. For this, the posting lists of all keys containing this term are merged to form a list of these documents. The result set, however, might become uncomfortably large.

In example 1, a user looking for information about the month of July would directly receive retrieval units from document1, if he were to execute a query q = {(July, /month)} or a query with a similar structure such as {(July, /calendar/month)}. He would receive document2 only if similarity between the indexed structure /name and the query structure /month is detected. This could be the case when using an ontology that indicates a connection between words like “name” and “month-name”. If the user executes q = {July, Sasha}, and (Sasha,-) was an additional key with posting list = {document4}, the retrieval component would not find a direct key for q. The query would therefore be split into sets of query tuples, and those be checked. For (Sasha,-), retrieval units from document4 would be returned if they are sufficiently relevant to the entire query q. For (July, -), which is not a key since its frequency at indexing time exceeded threshold PL_{max}, the user would be requested to choose a more specific expression by adding a more specific structure to the term or amending it with more terms. The user could choose from the following list: {structure of July: /name, /month; amending terms: apple, Peter}. Depending on the user’s choice, retrieval units from either document1 or document2 would be returned. If the user does not make a choice, either “July” will be deleted from q, or retrieval units from both document1 and document2 will be returned. If the set of specifying structures or amending terms recommended to the user is too large, it can be reduced by using clustering, filtering or caching strategies.

4. Future work

We are currently in the process of implementing SPIRIX, a search engine for P2P information retrieval of XML-documents that uses the proposed techniques and algorithms. Experiments will concentrate on evaluating several computation alternatives for the ranking process. We will compare different ranking formulas by evaluating with the INEX test collection. Moreover, we are going to analyze theoretically estimated and experimentally confirmed performance measurements such as communication overhead, storage and bandwidth consumptions as well as response and indexing times.

For the underlying P2P-layer, an adapted Kademlia-variant is applied that we currently develop in order to optimize it for XML Information Retrieval. Furthermore, the developed P2P protocol will collect peer statistics such as response time and latency so that a peer score can be computed and influence the decisions, which peers will participate in answering a given query.

Executing queries involves splitting the query in existing keys that cover all query tuples, and requesting key and tuple frequencies from the P2P-network. Consequently, this task needs to be optimized in terms of amount and size of messages sent between peers. Appropriate filtering methods and a well chosen caching strategy can be developed to this end.

Future work will also include further parallelization of the ranking algorithm. In addition, we plan to parallelize the indexing that has previously used distributed information but is performed sequentially.

5. References

[1] Aberer, K.; Klemm, F.; Rajman, M.; Wu, J.: ***An Architecture for Peer-to-Peer Information Retrieva***l. Workshop on Peer-to-Peer Information Retrieval, 2004.

[2] Amer-Yahia, S.; Lalmas, M.: ***XML Search: Languages, INEX and Scoring***. SIGMOD RecVol. 35, No. 4, 2006.

[3] Balakrishnan, H; Kaashoek, F.; Karger, D.; Morris, R.; Stoica, I.: ***Looking Up Data in P2P Systems***. Communications of the ACM, Vol. 46, No. 2, 2003.

[4] El-Ansary, Sameh; Haridi, Seif: ***An Overview of Structured Overlay Networks.*** In: Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks, CRC Press, 2005.

[5] Frieder, O.; Grossmann, D.: ***Information Retrieval. Algorithms and Heuristics.*** 2nd Ed., Springer, 2004.

[6] Malik, S.; Trotman, A.; Lalmas, M.; Fuhr, N.: ***Overview of INEX 2006.*** In: Proc. of the Fifth Workshop of INEX, Dagstuhl, Germany, 2007.

[7] Maymounkov, P.; Mazieres, D.: ***Kademlia: A peer-to-peer information system based on the xor metric***. In Proceedings of IPTPS02, Cambridge, USA, 2002.

[8] Pass, G.; Chowdhury, Abdur; Torgeson, Cayley:
A Picture of Search. First International Conference on Scalable Information Systems, Hong Kong, 2006.

[9] Podnar, I.; Luu, T.; Rajman, M.; Klemm, F.; Aberer, K.:
A Peer-to-Peer Architecture for Information Retrieval Across Digital Library Collections. ECDL'06, Alicante, Spain, 2006.

[10] Risson, J.; Moors, T.: ***Survey of research towards robust peer-to-peer networks – search methods***. Technical Report UNSW-EE-P2P-1-1, Australia, 2004.

[11] Robertson, S.; Zaragoza, H.; Taylor, M.: ***Simple BM25 extension to multiple weighted fields***. In: Proc. of CIKM'04, ACM Press, New York, USA, 2004.

[12] Steinmetz, R.; Wehrle, K. (eds.): ***Peer-to-Peer Systems and Applications.*** LNCS No. 3485, Springer, 2005.

[13] Tang, C.; Xu, Z.; Dwarkadas, S.: ***Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks***. In: Proc. of SIGCOMM'03, Karlsruhe, Germany, 2003.

[14] Winter, J.; Drobnik, O.: ***An Architecture for XML Information Retrieval in a Peer-to-Peer Environment.*** ACM PIKM2007 at ACM 16th Conference on Information and Knowledge Management (CIKM 2007), Lisbon, Portugal, 2007.

[15] World Wide Web Consortium: ***Extensible Markup Language (XML) 1.1 (Second Edition)***. W3C Recommendation, 16. Aug. 2006, http://www.w3.org/TR /2006 /REC-xml11-20060816/.

[16] World Wide Web Consortium: ***XML Path Language (XPath) Version 1.0.*** W3C Recommendation, 16. Nov. 1999, http://www.w3.org/TR/xpath.

Authors

Judith Winter is a research assistant in the Institute of Computer Science, Group of Telematics, at the J.W.Goethe-University, Frankfurt, Germany. Her main research interests include Peer-to-Peer systems, distributed Information Retrieval, and XML Information Retrieval. The focus of her PhD thesis is to develop a P2P search engine for XML Information Retrieval.
http://www.tm.informatik.uni-frankfurt.de/winter

Oswald Drobnik is a full professor in the Institute of Computer Science and head of the Telematics Group at the J.W.Goethe-University, Frankfurt, Germany, since 1988. From 1981-1988 he was professor with Karlsruhe University, Germany. Current research interests include distributed and parallel systems, information retrieval, and mobile computing.
http://www.tm.informatik.uni-frankfurt.de/Plone/Mitarbeiter/drobnik