Towards Collaborative Search in Digital Libraries Using Peer-to-Peer Technology

Matthias Bender, Sebastian Michel, Christian Zimmer, Gerhard Weikum
{mbender, smichel, czimmer, weikum}@mpi-sb.mpg.de

Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany

Abstract. We consider the problem of collaborative search across a large number of digital libraries and query routing strategies in a peer-to-peer (P2P) environment. Both digital libraries and users are equally viewed as peers and, thus, as part of the P2P network. Our system provides a versatile platform for a scalable search engine combining local index structures of autonomous peers with a global directory based on a distributed hash table (DHT) as an overlay network.

1 Introduction

The peer-to-peer (P2P) approach, which has become popular in the context of file-sharing systems such as Gnutella or KaZaA, allows to handle huge amounts of data in a distributed way. In such a system, all peers are equal and all of the functionality is shared among all peers so that there is no single point of failure and the load is balanced across a large number of peers. These characteristics offer potential benefits for building a powerful search engine in terms of scalability, resilience to failures, and high dynamics. In addition, a P2P search engine can potentially benefit from the intellectual input of a large user community, for example, prior usage statistics, personal bookmarks, or implicit feedback derived from user logs and click streams.

Our framework combines well-studied search strategies with new aspects of P2P routing strategies. In our context of digital libraries, a peer can either be a library itself or a user that wants to benefit from the huge amount of data in the network. Each peer is a priori autonomous and has its own local search engine with a crawler and a corresponding local index. Peers share their local indexes (or specific fragments of local indexes) by posting the meta-information into the P2P network, thus effectively forming a large global, but completely decentralized directory. In our approach, this directory is maintained as a distributed hash table (DHT). A query posed by a user is first executed on the user's own peer, but can be forwarded to other peers for better result quality. Collaborative search strategies use the global directory to identify peers that are most likely to hold relevant results. The query is then forwarded to an appropriately selected subset of these peers, and the local results obtained from there are merged by the query initiator.

2 Related Work

Recent research on P2P systems, such as Chord [25], CAN [22], Pastry [24], or P-Grid [1], is based on various forms of distributed hash tables (DHTs) and supports mappings from keys, e.g., titles or authors, to locations in a decentralized manner such that routing scales well with the number of peers in the system. Typically, an exact-match key lookup can be routed to the proper peer(s) in at most $O(log\ n)$ hops, and no peer needs to maintain more than $O(log\ n)$ routing information. These architectures can also cope well with failures and the high dynamics of a P2P system as peers join or leave the system at a high rate and in an unpredictable manner. Earlier work on scalable distributed storage structures, e.g., [16, 28], addressed similar issues. However, in all these approaches searching is limited to exact-match queries on keys. This is insufficient for text queries that consist of a variable number of keywords, and it is absolutely inappropriate when queries should return a ranked result list of the most relevant approximate matches [6]. Our work makes use of one of these systems, namely Chord, for efficiently organizing a distributed global directory; our search engine is layered on top of this basic functionality.

PlanetP [10] is a publish-subscribe service for P2P communities and the first system supporting content ranking search. PlanetP distinguishes local indexes and a global index to describe all peers and their shared information. The global index is replicated using a gossiping algorithm. The system, however, is limited to a few thousand peers.

Odissea [26] assumes a two-layered search engine architecture with a global index structure distributed over the nodes in the system. A single node holds the entire index for a particular text term (i.e., keyword or word stem). Query execution uses a distributed version of Fagin's threshold algorithm [11]. The system appears to cause high network traffic when posting document metadata into the network, and the query execution method presented currently seems limited to queries with one or two keywords only.

The system outlined in [23] uses a fully distributed inverted text index, in which every participant is responsible for a specific subset of terms and manages the respective index structures. Particular emphasis is put on three techniques to minimize the bandwidth used during multi-leyword searches: Bloom filters [3], caching, and incremental result gathering. Bloom filters are a compact representation of membership in a set, eliminating the need to send entire index lists across servers. Caching reduces the frequency of exchanging Bloom filters between servers. Incremental result gathering allows search operations to halt after finding a certain number of results.

[18] considers content-based retrieval in hybrid P2P networks where a peer can either be a simple node or a directory node. Directory nodes serve as super-peers, which may possibly limit the scalability and self-organization of the overall system. The peer selection for forwarding queries is based on the Kullback-Leibler divergence between peer-specific statistical models of term distributions. The approach that we propose in this paper also uses such statistical measures but applies them in a much more light-weight manner for better scalability,

primarily using bookmarks rather than full index information and building on a completely decentralized directory for meta-information.

Strategies for P2P request routing beyond simple key lookups but without considerations on ranked retrieval have been discussed in [30, 8, 7], but are not directly applicable to our setting. The construction of semantic overlay networks is addressed in [17, 9] using clustering and classification techniques; these techniques would be orthogonal to our approach. [27] distributes a global index onto peers using LSI dimensions and the CAN distributed hash table. In this approach peers give up their autonomy and must collaborate for queries whose dimensions are spread across different peers.

In addition to this recent work on P2P Web search, prior research on distributed IR and metasearch engines is potentially relevant, too. [4] gives an overview of algorithms for distributed IR like result merging and database content discovery. [12] presents a formal decision model for database selection in networked IR. [21] investigates different quality measures for database selection. [13, 19] study scalability issues for a distributed term index. GlOSS [14] and CORI [5] are the most prominent distributed IR systems, but neither of them aimed at very-large-scale, highly dynamic, self-organizing P2P environments (which were not an issue at the time these systems were developed).

A good overview of metasearch techniques is given by [20]. [29] discusses specific strategies to determine potentially useful local search engines for a given user query. Notwithstanding the relevance of this prior work, collaborative P2P search is substantially more challenging than metasearch or distributed IR over a small federation of sources such as digital libraries, as these approaches mediate only a small and rather static set of underlying engines, as opposed to the high dynamics of a P2P system.

3 Chord - A Scalable P2P Lookup Service

The efficient location of nodes in a P2P architecture is a fundamental problem that has been tackled from various directions. Early (but nevertheless popular) systems like Gnutella or KaZaA rely on unstructured architectures in which a peer forwards messages to all known neighbors. Typically, these messages include a Time-to-live (TTL) tag that is decreased whenever the message is forwarded to another peer. Even though studies show that this *message flooding* (or *gossiping*) works remarkably well in most cases, there are no guarantees that all relevant nodes will eventually be reached. Additionally, the fact that numerous unnecessary messages are sent interferes with our goal of a highly scalable architecture.

Chord [25] is a distributed lookup protocol that addresses this problem. It provides the functionality of a distributed hash table (DHT) by supporting the following *lookup* operation: given a key, it maps the key onto a node. For this purpose, Chord uses consistent hashing [15]. Consistent hashing tends to balance load, since each node receives roughly the same number of keys. Moreover, this load balancing works even in the presence of a dynamically changing hash range, i.e., when nodes fail or leave the system or when new nodes join.

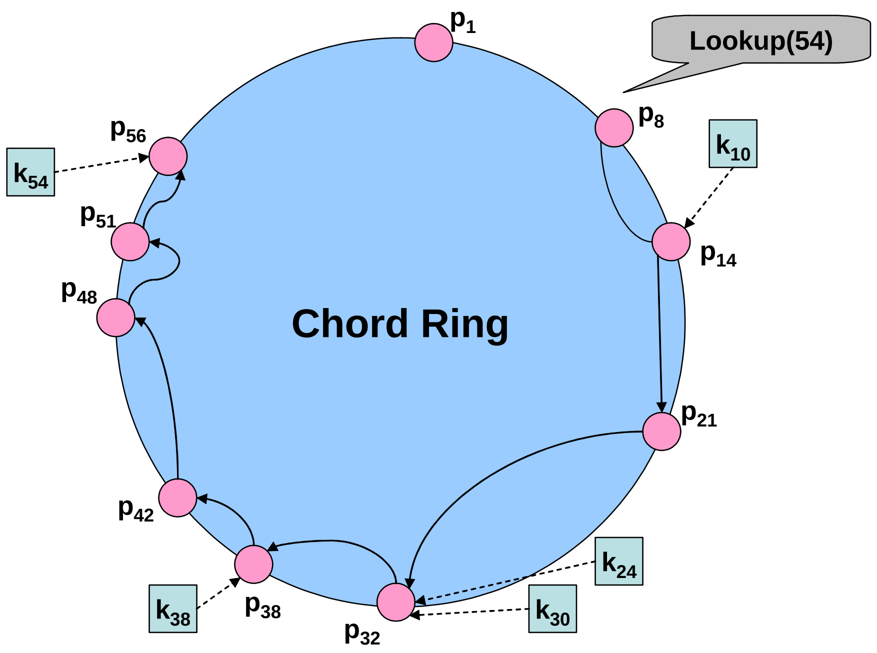
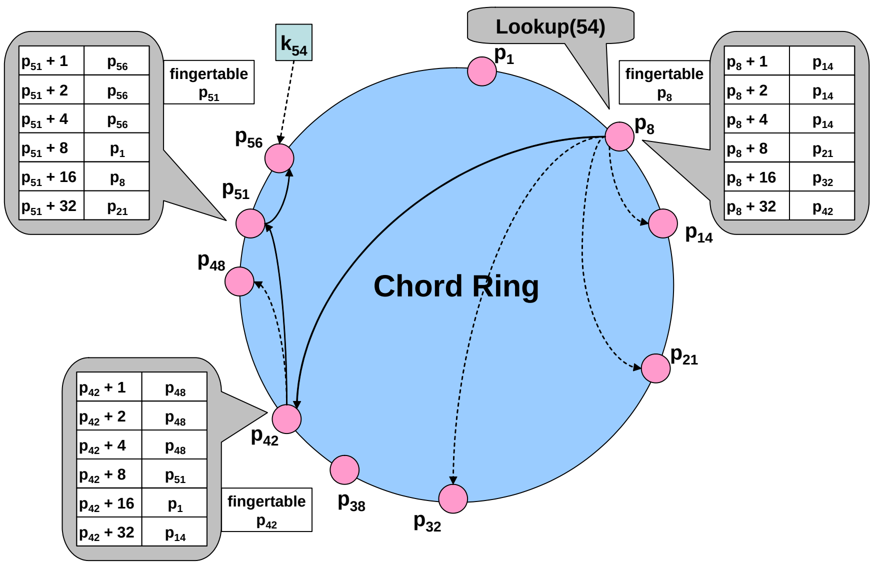


Figure 1. Chord Architecture

Chord not only gurarantees to find the node responsible for a given key, but also can do this very efficiently: in an N-node steady-state system, each node maintains information about only $O(\log N)$ other nodes, and resolves all lookups via $O(\log N)$ messages to other nodes. These properties offer the potential for efficient large-scale systems.

The intuitive concept behind Chord is as follows: all nodes p_i and all keys k_i are mapped onto the same cyclic ID space. In the following, we use keys and peer numbers as if the hash function had already been applied, but we do not explicitly show the hash function for simpler presentation. Every key k_i is now assigned to its closest successor p_i in the ID space, i.e. every node is responsible for all keys with identifiers between the ID of its predecessor node and its own ID.

For example, consider Figure 1. Ten nodes are distributed across the ID space. Key k_{54}, for example, is assigned to node p_{56} as its closest successor node. A naive approach of locating the peer responsible for the key is also illustrated: since every peer knows how to contact its current successor on the ID circle, a query for a key k_{54} initiated by peer p_8 is passed around the circle until it encounters a pair of nodes that straddle the desired identifier; the second in the pair (p_{56}) is the node that is responsible for the key. This lookup process closely resembles searching a linear list and has an expected number of hops of $O(N)$ to find a target node, while only requiring $O(1)$ information about other nodes.



p_{51} + 1	p_{56}
p_{51} + 2	p_{56}
p_{51} + 4	p_{56}
p_{51} + 8	p_1
p_{51} + 16	p_8
p_{51} + 32	p_{21}

p_8 + 1	p_{14}
p_8 + 2	p_{14}
p_8 + 4	p_{14}
p_8 + 8	p_{21}
p_8 + 16	p_{32}
p_8 + 32	p_{42}

p_{42} + 1	p_{48}
p_{42} + 2	p_{48}
p_{42} + 4	p_{48}
p_{42} + 8	p_{51}
p_{42} + 16	p_1
p_{42} + 32	p_{14}

Figure 2. Scalabe Lookups Using Finger Tables

To accelerate lookups, Chord maintains additional routing information: each peer p_i maintains a routing table called *finger table*. The m-th entry in the table of node p_i contains a pointer to the first node p_j that succeeds p_i by at least 2^{m-1} on the identifier circle. This scheme has two important characteristics. First, each node stores information about only a small number of other nodes, and knows more about nodes closely following it on the identifier circle than about nodes farther away. Secondly, a node's finger table does not necessarily contain enough information to *directly* determine the node responsible for an arbitrary key k_i. However, since each peer has finger entries at power of two intervals around the identifier circle, each node can forward a query at least halfway along the remaining distance between itself and the target node. This property is illustrated in Figure 2 for node p_8. It follows that the number of nodes to be contacted (and thus the number of messages to be sent) to find a target node in an N-node system is $O(\log N)$.

Chord implements a stabilization protocol that each peers runs periodically in the background and which updates Chord's finger tables and successor pointers in order to ensure that lookups execute correctly as the set of participating peers changes. But even with routing information becoming stale, system performance degrades gracefully. Chord can also guarantee correct lookups if only one piece of information per node is correct.

Chord can provide lookup services for various applications, such as distributed file systems or cooperative mirroring. However, Chord by itself is not a search engine, as it only supports single-term exact-match queries and does not support any form of ranking.

4 Design Fundamentals

Figure 3 illustrates our new approach which closely follows a publish-subscribe paradigm. We view every library as autonomous. Peers, i.e. libraries acting as peers, can post meta-information at their discretion. Our conceptually global but physically distributed directory does not hold information about individual documents previously crawled by the peers, but only very compact aggregated information about the peers' local indexes and only to the extent that the individual peers are willing to disclose to other peers. We use a distributed hash table (DHT) to partition the term space, such that every peer is responsible for a randomized subset of terms within the global directory. For failure resilience and availability, the entry for a term may be replicated across multiple peers.

Every peer publishes a summary (*Post*) for every term in its local index to the underlying overlay network. A Post is routed to the peer currently responsible for the Post's term. This peer maintains a *PeerList* of all postings for this term from across the network. Posts contain contact information about the peer who posted this summary together with local IR-style statistics (e.g., TF and IDF values [6]) for a term and other quality-of-service measures (e.g., length of the index list for a given term, or average response time for remote queries).

Users wishing to pose a query are equally modelled as peers. Their potential input to the global directory consists of local bookmarks that conceptually represent high-authority documents within the overall document space.

The querying process for a multi-term query proceeds as follows: First, the querying peer retrieves a list of potentially useful libraries by issuing a *PeerList request* for each query term to the global directory. Next, a number of promising libraries for the complete query is selected from these PeerLists (e.g., based on the quality-of-service measures associated with the Posts). Subsequently, the query is forwarded to these carefully selected libraries and executed based on the their local indexes. Note that this communication is done in a pairwise point-to-point manner between the peers, allowing for efficient communication and limiting the load on the global directory. Finally, the results from the various libraries are combined at the querying peer into a single result list.

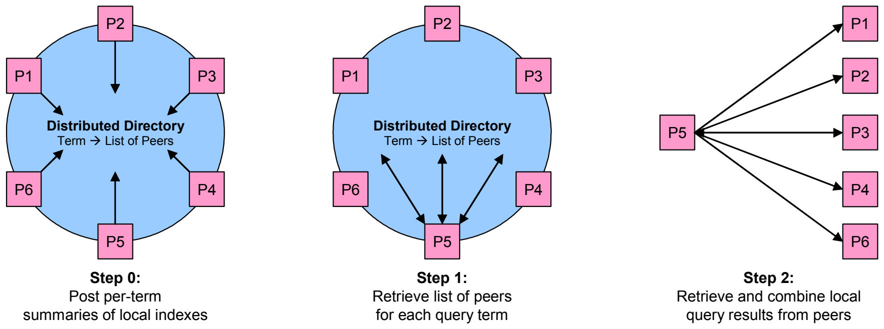


Figure 3. P2P Query Routing

We have chosen this approach for the following reasons:

- The goal of finding high-quality search results with respect to precision and recall cannot easily be reconciled with the design goal of unlimited scalability, as the best information retrieval techniques for query execution rely on large amounts of document metadata. In contrast, posting only aggregated information about local indexes and executing queries at carefully selected peers exploits extensive local indexes for good query results while, at the same time, limiting the size of the global directory and, thus, consuming only little network bandwidth.
- If each peer were to post metadata about each and every document it has crawled, the amount of data moved across the network and, thus, the amount of data held by the distributed directory would increase drastically as more and more peers enter the network. In contrast, our design allows each peer to publish merely a concise summary per term representing its local index. As new peers enter the network, we expect this approach to scale very well as more and more peers jointly maintain this moderately growing global directory.

This approach can easily be extended in a way that multiple distributed directories are created to store information beyond local index summaries, such as information about user bookmarks or relevance assessments derived from peer-specific query logs, click streams, or explicit user feedback. This information could be leveraged when executing a query to further enhance result quality.

5 System Model

In this section we formalize the design that we have previouly presented. Let $P := \{p_i | 1 \leq i \leq r\}$ be the set of peers currently attached to the system. Let $D := \{d_i | 1 \leq i \leq n\}$ be the global set of all documents; let $T := \{t_i | 1 \leq i \leq m\}$ analogously be the set of all terms.

Each peer $p_i \in P$ has one or more of the following local data available:

- Local index lists for terms in $T_i \subseteq T$ (usually $|T_i| \ll |T|$).
  The local index lists cover all terms in the set of locally seen documents $D_i \subseteq D$ (usually $|D_i| \ll |D|$).
- Bookmarks $B_i \subseteq D_i$ ($|B_i| \ll |D|$)
  Bookmarks are intellectually selected links to selected documents or other peer profile information and, thus, are a valuable source for high-quality search results as well as for the thematic classification of peers.
- Cached documents $C_i \subseteq D$
  Cached documents are readily available from a peer.

Separate hash functions $hash_{terms} : T \rightarrow ID$, $hash_{bookmarks} : D \rightarrow ID$, and $hash_{cached} : D \rightarrow ID$ can be used in order to build conceptually global, but physically distributed directories that are well-balanced across the peers in the ID space.

Given hash functions that assign identifiers to keys using $id_{k,j} := hash_j(k)$ with $j \in \{terms, bookmarks, ...\}$, the underlying distributed hash table offers a function $lookup : ID \rightarrow P$ that returns the peer p currently responsible for an id.

Building on top of this basic functionality, different PeerList requests plr_j can be defined as functions $plr_{terms} : T \times P \rightarrow 2^P$, $plr_{bookmarks} : D \times P \rightarrow 2^P$, and $plr_{cache} : D \times P \rightarrow 2^P$ that, from a peer p previously determinded using *lookup*, return lists of peers that have posted information about a key id in dimension j. Note that $id_{k,j}$ for a specific key k is unambiguously defined across the directory using $hash_j$(k).

In order to form a distributed directory, each peer p_i at its own discretion globally posts subsets $T'_i \subseteq T_i$, $B'_i \subseteq B_i$, and $C'_i \subseteq C_i \subseteq D$

(potentially along with further information or local QoS statistics) forming the corresponding global directories:

– $systerms : T \rightarrow 2^P$ with $systerms(t) = plr_{terms}(t, lookup(hash_{terms}(t)))$
  This directory provides a mapping from terms to PeerLists and can be used to identify candidate peers that hold index information about a specific term.
– $sysbm : D \rightarrow 2^P$ with $sysbm(d) = plr_{bookmarks}(d, lookup(hash_{bookmarks}(d)))$
  This function provides information about which peers have bookmarked specific documents and is a combination of the above methods analogously to *systerms*.
– $syscd : D \rightarrow 2^P$ with $syscd(d) = plr_{cached}(d, lookup(hash_{cached}(d)))$
  This function provides information about the availability of documents in the caches of local peers, which is a valuable information for the efficient gathering of results.

We consider a query q as a set of $(term, weight)$-pairs and the set of available queries as $Q := 2^{T \times \mathbb{R}}$. In order to process a query q, first a candidate set of peers that are confronted with the query has to be determined. This can be done using the functions $selection_{terms} : Q \rightarrow 2^P$, $selection_{bookmarks} : 2^D \rightarrow 2^P$, and $selection_{cached} : 2^D \rightarrow 2^P$ that select candidate subsets for each dimension by appropriately combining the results returned by *systerms*, *sysbm*, and *syscd*, respectively. These candidate subsets are combined (e.g., by intersection or union) using a function $comb : 2^P \times 2^P \times 2^P \rightarrow 2^P$.

Combining the above, the final candidate set is computed using a function

$$selection : Q \times 2^D \times 2^D \rightarrow 2^P$$

$$selection(q, B''_0, C''_0) := comb(select_{terms}(q), select_{bookmarks}(B''_0), select_{cached}(C''_0))$$

where $B''_0 \subseteq B_0$ and $C''_0 \subseteq C_0$ are the bookmarks and cached documents, respectively, that the querying peer has chosend to support query execution. For example, a peer may choose its own bookmarks and a sample of its cached documents as B''_0 and C''_0, respectively.

The execution of a query is a function $exec : 2^P \times Q \rightarrow 2^D$ that combines the local results returned by the peers that are involved in the query execution into one single final result set. Finally, we can define the global query execution function $result : Q \times 2^D \times 2^D \rightarrow 2^D$ that is evaluated as

$$result(q, B''_0, C''_0) := exec(selection(q, B''_0, C''_0), q)$$

$$= exec(comb(select_{terms}(q), select_{bookmarks}(B''_0), select_{cached}(C''_0)), q)$$

6 Implementation

Figure 4 illustrates the architecture of a single library peer as part of our distributed system. Each peer works on top of our globally distributed index which is organized as a distributed hash table (DHT) that provides a mapping from terms to peers by returning a *PeerDescriptor* object representing the peer currently responsible for a term. A *Communicator* can be established to send messages to other peers. Every peer has an *Event Handler* that receives incoming messages and forwards them to the appropriate local components.

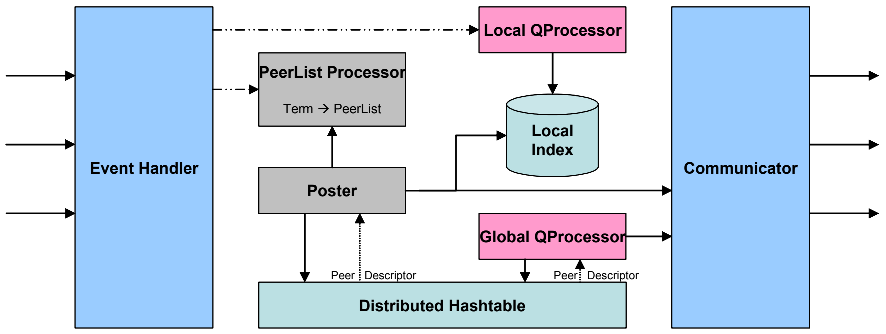
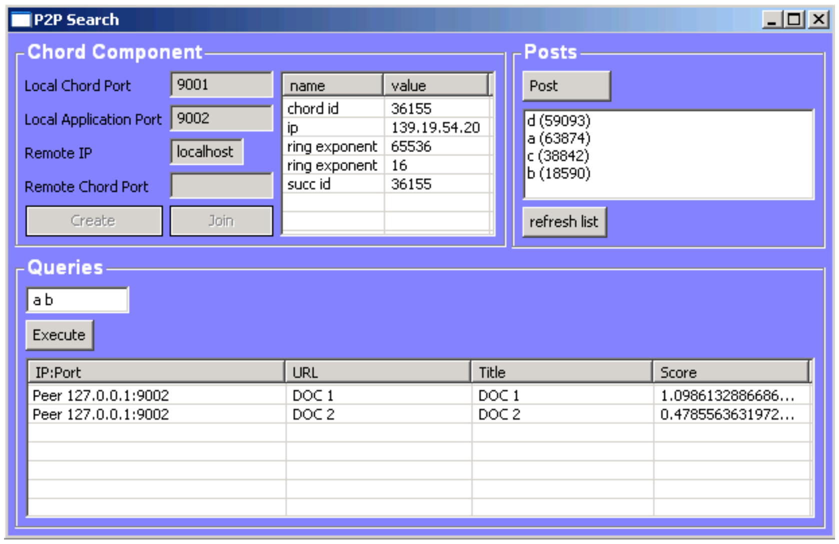


Figure 4. System Architecture

Every peer has its own local index that can be imported from external crawlers and indexers. The index is used by the *Local QueryProcessor* component to answer queries locally and by the *Poster* component to publish per-term summaries (*Posts)* to the global directory. To do so, the Poster uses the underlying DHT to find the peer currently responsible for a term; the *PeerList Processor* at this peer maintains a PeerList of all Posts for this term from across the network. When the user poses a query, the *Global QueryProcessor* component analogously uses the DHT to find the peer responsible for each query term and retrieves the respective PeerLists from the PeerList Processors using Communicator components. After appropriately processing these lists, the Global QueryProcessor forwards the complete query to selected peers, which in turn process the query using their Local QueryProcessors and return their results. Finally, the Global QueryProcessor merges these results and presents them to the user.

We have built a prototype system that handles the above procedures. Our system uses a Java-based reimplementation of Chord [25] as its underlying DHT, but can easily be used with other DHT's providing a *lookup*(*key*) method. Communication is conducted socket-based, but Web-Service-based [2] peers can easily be included to support an arbitrarily heterogeneous environment. The local index is stored in a database. It consists of a collection of standard IR measures, such as TF and IDF values. Result ranking is based on a smoothed TF*IDF quality measure. Figure 5 shows a screenshot of the user interface of our prototype. The user creates a peer by either creating a new Chord ring or by joining an existing system. Both actions require the specification of a local Chord port for communication concerning the global directory and a local application port for direct peer-to-peer communication. The join operation requires additional information on how to find an already existing peer. Status information regarding the Chord ring is displayed. The Posts section provides information about the terms that a peer is currently responsible for, i.e., for which it has received Posts from other peers. The button *Post* posts the information contained in the local index to the DHT. The Queries section can be used to execute queries. Similar to Google, multiple keywords can be entered into a form field. After query execution, the results obtained from the system are displayed.



Local Chord Port	9001
Local Application Port	9002
Remote IP	localhost
Remote Chord Port	

name	value
chord id	36155
ip	139.19.54.20
ring exponent	65536
ring exponent	16
succ id	36155

d (59093)
a (63874)
c (38842)
b (18590)

a b

IP:Port	URL	Title	Score
Peer 127.0.0.1:9002	DOC 1	DOC 1	1.0986132886686...
Peer 127.0.0.1:9002	DOC 2	DOC 2	0.4785563631972...

Figure 5. Prototype GUI

7 Ongoing and Future Work

Our prototype implementations allows for the easy exchange of strategies for query routing (i.e., selecting the peers to which the query is sent) as well as for merging the results returned by different peers. We are currently analyzing different strategies and are preparing extensive comparative experiments. We want to contact as few peers as possible to retrieve the best possible results, i.e., we want to estimate a $benefit/cost$ ratio when deciding on whether to contact a specific peer. While a typical cost measure could be based on expected response time (network latency, current load of remote peer), meaningful benefit measures seem harder to find. Possible measures could follow the intuition that good answers are expected to come from peers that are similar to the query, but at the same time have only little overlap with our local index and, thus, can potentially contribute new results. We also take a closer look at existing strategies for combining local query results from metasearch engine research and try to fit those with our P2P environment.

We investigate the trade-offs of not storing *all* Posts for a term, but only the top-k posts (based on some quality measure) to reduce space consumption of the global directory. While this seems intuitive at first sight (good results should come from good peers), early experiments indicate that this strategy might be dangerous for multi-term queries, as good combined results are not necessarily top results for any one of the search terms.

Due to the dynamics typical for P2P systems, Posts stored in the PeerLists become invalid (peers may no longer be accessible, or the responsibility for a specific term may have moved to another peer). A possible mechanism to handle these problems is to assign a TTL (Time-to-live) stamp to every Post in the list. Every peer periodically revalidates its Posts. Stale Posts will eventually be removed from the PeerList. We address the question of choosing a good time period for refreshing the Posts and compare this strategy to a strategy of actively moving Posts to other peers as responsibilities change.

References

1. K. Aberer, M. Punceva, M. Hauswirth, and R. Schmidt. Improving data access in p2p systems. *IEEE Internet Computing*, 6(1):58–67, 2002.
2. G. Alonso, F. Casati, and H. Kuno. *Web Services - Concepts, Architectures and Applications*. Springer, Berlin;Heidelberg;New York, 2004.
3. B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
4. J. Callan. Distributed information retrieval. *Advances in information retrieval, Kluwer Academic Publishers.*, pages 127–150, 2000.
5. J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 21–28. ACM Press, 1995.
6. S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann, San Francisco, 2002.
7. E. Cohen, A. Fiat, and H. Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. In *Proceedings of the IEEE INFOCOM'03 Conference, April 2003*, April 2003.
8. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. of the 28th Conference on Distributed Computing Systems*, July 2002.
9. A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. Technical report, Stanford University, October 2002.
10. F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. Technical Report DCS-TR-487, Rutgers University, Sept. 2002.
11. R. Fagin. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.*, 58(1):83–99, 1999.
12. N. Fuhr. A decision-theoretic approach to database selection in networked IR. *ACM Transactions on Information Systems*, 17(3):229–249, 1999.
13. T. Grabs, K. Böhm, and H.-J. Schek. Powerdb-ir: information retrieval on top of a database cluster. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 411–418. ACM Press, 2001.

14. L. Gravano, H. Garcia-Molina, and A. Tomasic. Gloss: text-source discovery over the internet. *ACM Trans. Database Syst.*, 24(2):229–264, 1999.
15. D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.
16. W. Litwin, M.-A. Neimat, and D. A. Schneider. Lh* – a scalable, distributed data structure. *ACM Trans. Database Syst.*, 21(4):480–525, 1996.
17. A. Löser, F. Naumann, W. Siberski, W. Nejdl, and U. Thaden. Semantic overlay clusters within super-peer networks. In *Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing, 2003 (DBISP2P 03)*, pages 33–47.
18. J. Lu and J. Callan. Content-based retrieval in hybrid peer-to-peer networks. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 199–206. ACM Press, 2003.
19. S. Melnik, S. Raghavan, B. Yang, and H. Garcia-Molina. Building a distributed full-text index for the web. *ACM Trans. Inf. Syst.*, 19(3):217–241, 2001.
20. W. Meng, C. T. Yu, and K.-L. Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89, 2002.
21. H. Nottelmann and N. Fuhr. Evaluating different methods of estimating retrieval quality for resource selection. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 290–297. ACM Press, 2003.
22. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM 2001*, pages 161–172. ACM Press, 2001.
23. P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of International Middleware Conference*, pages 21–40, June 2003.
24. A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.
25. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.
26. T. Suel, C. Mathur, J. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasunderam. Odissea: A peer-to-peer architecture for scalable web search and information retrieval. Technical report, Polytechnic Univ., 2003.
27. C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 175–186. ACM Press, 2003.
28. R. Vingralek, Y. Breitbart, and G. Weikum. Snowball: Scalable storage on networks of workstations with balanced load. *Distributed and Parallel Databases*, 6(2):117–156, 1998.
29. Z. Wu, W. Meng, C. T. Yu, and Z. Li. Towards a highly-scalable and effective metasearch engine. In *World Wide Web*, pages 386–395, 2001.
30. B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 5–14. IEEE Computer Society, 2002.