The MINERVA[*] Project:
Database Selection in the Context of P2P Search

Matthias Bender, Sebastian Michel, Gerhard Weikum, Christian Zimmer
{mbender, smichel, weikum, czimmer}@mpi-sb.mpg.de
Max-Planck-Institut für Informatik, 66123 Saarbrücken

Abstract: This paper presents the MINERVA project that protoypes a distributed search engine based on P2P techniques. MINERVA is layered on top of a Chord-style overlay network and uses a powerful crawling, indexing, and search engine on every autonomous peer. We formalize our system model and identify the problem of efficiently selecting promising peers for a query as a pivotal issue. We revisit existing approaches to the database selection problem and adapt them to our system environment. Measurements are performed to compare different selection strategies using real-world data. The experiments show significant performance differences between the strategies and prove the importance of a judicious peer selection strategy. The experiments also present first evidence that a small number of carefully selected peers already provide the vast majority of all relevant results.

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 handling huge amounts of data in a distributed and self-organizing 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 evenly balanced across a large number of peers. These characteristics offer enormous potential benefits for search capabilities powerful in terms of scalability, efficiency, and resilience to failures and dynamics. Additionally, such a search engine can potentially benefit from the intellectual input (e.g., bookmarks, query logs, etc.) of a large user community. One of the key difficulties, however, is to efficiently select promising peers for a particular information need.

In spite of being a young paradigm, P2P exposes large overlap with traditional database research and can highly benefit from existing work. However, the peculiarities of such a distributed architecture require a different view on some aspects. For example, the absence of a centralized indexing facility together with the difficulties to calculate global metrics in this large and highly dynamic system hamper the use of traditional methods for database selection.

This paper presents the architecture of the MINERVA P2P Web search project and proposes and evaluates strategies for routing queries to peers. Each peer is considered 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 meta-information into the P2P network. This meta-information contains compact statistics

[*] Minerva is the Roman goddess of science, wisdom, and learning, and also happens to be a Greek underwear manufacturer.

and quality-of-service information, and effectively forms a global directory. However, this directory is implemented in a completely decentralized and largely self-organizing manner. More specifically, we maintain it as a distributed hash table (DHT) using the (re-implemented and adapted) algorithms of the Chord system [SMK$^+$01]. Our per-peer engine uses the global directory to identify candidate peers that are most likely to provide good query results. A query posed by a user is first executed on the user's own peer, but can additionally be forwarded to other peers for better result quality. The local results obtained from there are merged by the query initiator.

The novel contributions of our work lie in the system-oriented approach to P2P Web search. We present measurements of a fully operational system on real-life data, considering both the quality of search results and the overhead and run-time efficiency of the system. Novel aspects of the Minerva architecture are the way we leverage Chord-style overlay networks for efficient managing of the peers' postings on their index contents and statistical summaries, and the support for a broad variety of advanced peer selection strategies. Compared to earlier work on database selection in distributed information retrieval (e.g., [Fu99, CLC95, MYL02]) our approach has been explicitly designed for the large scale and high dynamics of P2P systems.

After presenting related work in Section 2, we present the P2P Lookup Service Chord in Section 3. Section 4 gives a short introduction on Information Retrieval basics necessary for the remainder of this paper. The system design is presented in Section 5 and formalized to a system model in Section 6. Section 7 discusses the implementation that serves as an experimental testbed for studying the different Peer Selection strategies introduced in Section 8. Section 9 shows early experimental results obtained from the prototype with real-world data. Future research directions conclude this paper in Section 10.

2 Related Work

Recent research on P2P systems, such as Chord [SMK$^+$01], CAN [RFH$^+$01], Pastry [RD01], P2P-Net [BB04], or P-Grid [APHS02] 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. However, the approaches are limited to exact-match, single keyword queries on keys. This is insufficient when queries should return a ranked result list of the most relevant approximate matches [Ch02].

In the following we briefly discuss some existing approaches towards P2P Web search. Galanx [WGdW03] is a peer-to-peer search engine implemented using the Apache HTTP server and BerkeleyDB. It directs user queries to relevant nodes by consulting local peer indexes similar to our approach.

PlanetP [CAPMN02] 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 [SMW$^+$03] 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 [Fa99]. 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 [RV03] 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-keyword searches.

[LC03] 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.

Strategies for P2P request routing beyond simple key lookups but without considerations on ranked retrieval have been discussed in [YGM02, CGM02a, CFK03], but are not directly applicable to our setting. The construction of semantic overlay networks is addressed in [LNS$^+$03, CGM02b] using clustering and classification techniques; these techniques would be orthogonal to our approach. [TXD03] 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. [ACMHP04] addresses the problem of building scalable semantic overlay networks and identifies strategies for their traversal.

In addition to this recent work on P2P Web search, prior research on distributed IR and metasearch engines is potentially relevant, too. [Ca00] gives an overview of algorithms for distributed IR style result merging and database content discovery. [Fu99] presents a formal decision model for database selection in networked IR. [NF03] investigates different quality measures for database selection. [GBS01, MRYGM01] study scalability issues for a distributed term index.

A good overview of metasearch techniques is given by [MYL02]. [WMYL01] 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, 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 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 [SMK$^+$01] 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 [KLL+97]. 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.

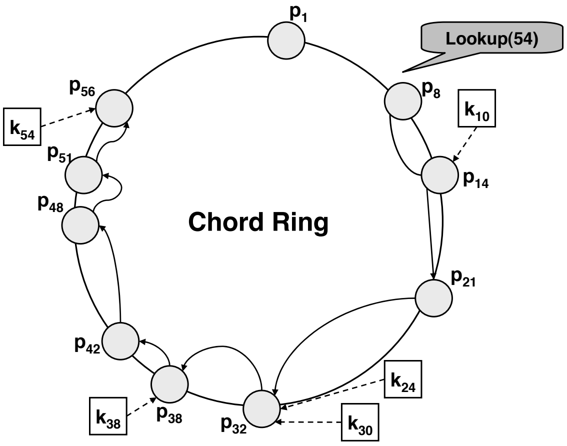


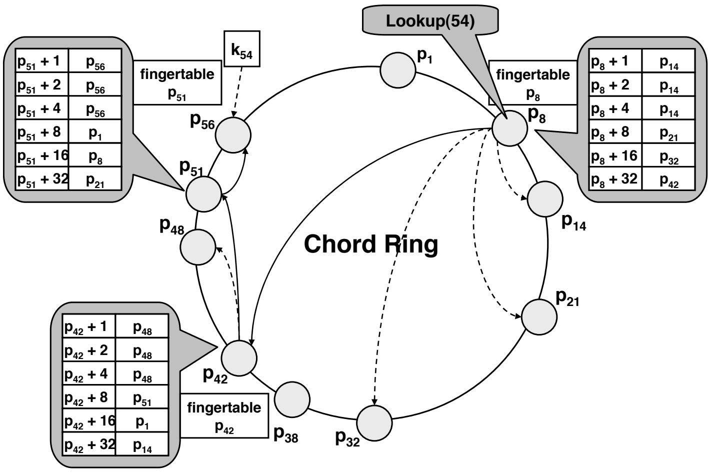
Figure 1: Chord Architecture

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 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 a key is also illustrated: since every peer knows how to contact its current successor on the ID circle, a query for 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 $O(N)$ hops to find a target node, while only requiring $O(1)$ information about other nodes.

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



$p_{51}+1$	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

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 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 Information Retrieval Basics

Information Retrieval (IR) systems keep large amounts of unstructured or weakly structured data, such as text documents or HTML pages, and offer search functionalities for delivering documents relevant to a query. Typical examples of IR systems include web search engines or digital libraries; in the recent past, relational database systems are integrating IR functionality as well.

The search functionality is typically accomplished by introducing measures of similarity between the query and the documents. For text-based IR with keyword queries, the similarity function typically takes into account the number of occurences and relative positions of each query term in a document. Section 4.1 explains the concept of *inverted index lists* that support an efficient query execution and section 4.2 introduces one of the most popular similarity measures, the so-called *TF*IDF* measure. For further reading, we refer the reader to [Ch02, MS99].

4.1 Inverted Index Lists

The concept of inverted index lists has been developed in order to efficiently identify those documents in the dataset that contain a specific query term. For this purpose, all terms that appear in the collection form a tree-like index structure (often a B^+-tree or a trie) where the leaves contain a list of unique document identifiers for all documents that contain this term (Figure 3). Conceptually, these lists are combined by intersection or union for all query terms to find candidate documents for a specific query. Depending on the exact query execution strategy, the lists of document identifiers may be ordered according to the document identifiers or according to a score value to allow efficient pruning.

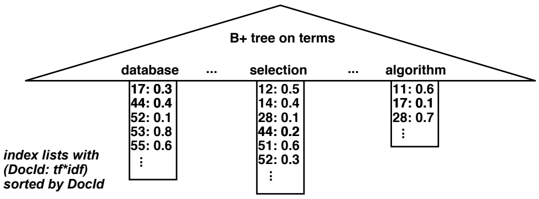


Figure 3: B+ Tree of Inverted Index Lists

4.2 $TF * IDF$ Measure

The number of occurences of a term t in a document d is called *term frequency* and typically denoted as $tf_{t,d}$. Intuitively, the significance of a document increases with the number of occurences of a query term. The number of documents in a collection that contain a term t is called *document frequency* (df_t); the *inverse document frequency* (idf_t) is defined as the inverse of df_t. Intuitively, the relative importance of a query term decreases as the number of documents that contain this term increases, i.e., the term offers less differentiation between the documents. In practice, these two measures may be normalized (e.g., to values between 0 and 1) and dampened using logarithms. A typical representative of this family of $tf * idf$ formulae that calculates the weight $w_{i,j}$ of the i-th term in the j-th document is

$$w_{i,j} := \frac{tf_{i,j}}{max_t\{tf_{t,j}\}} * log(\frac{N}{df_i})$$

where N is the total number of documents in the collection.

In recent years, other relevance measures based on statistical language models and probabilistic IR have received wide attention [Fu99, CL03]. For simplicity and because our focus is on P2P distributed search, we use the still most popular $tf * idf$ scoring family in this paper.

4.3 Top-k Index Processing

A good algorithm should avoid reading inverted index lists completely, but limit the effort to $O(k)$ where k is the number of desired results. In the IR and multimedia-search literature, various algorithms have been proposed to accomplish this. The best known general-purpose method for top-k queries is Fagin's threshold algorithm (TA) [FLN01], which has been independently proposed also by Nepal et al. [NR99] and Güntzer et al.

[GBK00]. It uses index lists that are sorted in descending order of term scores under the additional assumption that the final score for a document is calculated using a monotone aggregation function (such as a simple sum function). TA traverses all inverted index lists in a round-robin manner, i.e., lists are mainly traversed using sorted accesses. For every new document d encountered, TA uses random accesses to calculate the final score for d and keeps this information in a document candidate set. Since TA additionally keeps track of a higher bound for documents not yet encountered, the algorithm terminates as soon as this bound assures that no unseen document can enter the candidate set. Probabilistic methods have been studied in [TWS04] that can further improve the efficiency of index processing.

5 System Design

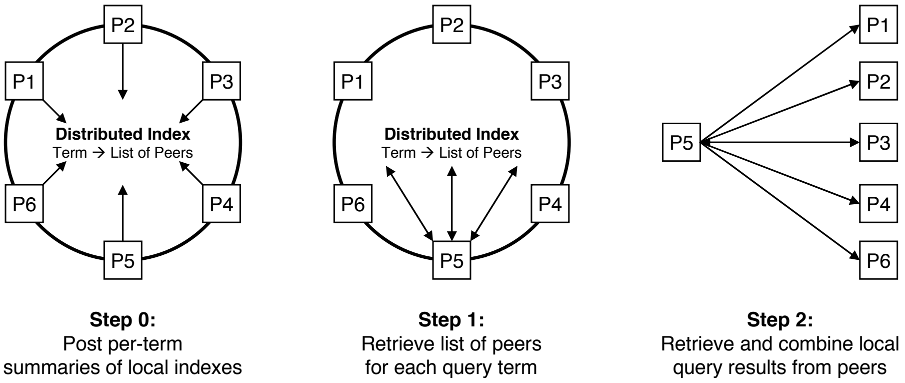
Figure 4 illustrates our approach which is layered on top of Chord and closely follows a publish-subscribe paradigm. We assume that every database forms a peer that is completely autonomous and has a local index that, e.g., can be imported from external crawlers and indexers. Our (conceptually global but physically distributed) directory holds only very compact, aggregated information about the peers' local indexes and only to the extent that the individual peers are willing to disclose. We use distributed hash tables to partition the term space, such that every peer is responsible for a randomized subset of terms within the global directory.

Every peer publishes a summary (*Post*) for every term in its local index to the underlying overlay network, which is routed to the peer currently responsible for this term. This peer maintains a *PeerList* of all postings for this term from across the network (for failure resilience and availability, the PeerLists may be replicated across multiple peers). Posts contain contact information about the peer who posted this summary together with statistics to calculate IR-style measures as introduced in Chapter 4 for a term and other information about the peer (e.g., index statistics, or quality-of-service measures like average response times).

The querying process for a multi-term query proceeds as follows: first, the querying peer retrieves a list of potentially useful peers by issuing a *PeerList request* for each query term to the underlying overlay network. Using database selection methods, a number of promising peers for the complete query is computed from these PeerLists (*peer selection*). Subsequently, the query is forwarded to these peers 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 peers are combined at the querying peer into a single result list (*Result Merging*). Chapter 8 gives details about different peer selection strategies.

The goal of finding high-quality search results with respect to precision and recall cannot be easily reconciled with the design goal of unlimited scalability, as the best information retrieval techniques for query execution rely on large amounts of document metadata. Posting only compact, aggregated information about local indexes and using database selection methods to limit the number of peers that actually execute a query limits the size of the global directory and reduces network traffic. We expect this approach to scale very well as more and more peers jointly maintain this moderately growing global directory.

The 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 local bookmarks [BMWZ04], information about relevance assessments (e.g., derived from peer-



Step 0:
Post per-term
summaries of local indexes

Step 1:
Retrieve list of peers
for each query term

Step 2:
Retrieve and combine local
query results from peers

Figure 4: P2P Query Routing

specific query logs or click streams), or explicit user feedback. This information could be leveraged when executing a query to further enhance result quality.

6 System Model

In this section we formalize our design. Let $P := \{p_i | 1 \leq i \leq r\}$ be the set of peers currently connected 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 has a local index for terms in $T_i \subseteq T$ (usually $|T_i| \ll |T|$). The local index contains IR-style statistics $s_t \in S$ for each term t in the set of indexed documents $D_i \subseteq D$ (usually $|D_i| \ll |D|$).

A hash function $hash : T \rightarrow ID$ is used in order to distribute terms across the available peers by assigning identifiers to terms. The underlying distributed hash table offers a function $lookup : ID \rightarrow P$ that returns the peer p currently responsible for an identifier.

A PeerList request plr to a peer p_i about a term t can now be defined as a function $plr : T \times P \rightarrow 2^{P \times S}$, that returns a list of peers that have posted statistics about the term t to peer p_i. The function call $lookup(hash(t))$ is used to determine which peer is responsible for hosting statistics about a term $t \in T$.

In order to form the distributed directory, each peer p_i posts s_t for all $t \in T_i' \subseteq T_i$ (T_i' is the subset of T_i that the peer p_i can select at its own discretion) forming the global directory:

$$systerms : T \rightarrow 2^{P \times S}$$

$$systerms(t) := plr(t, lookup(hash(t)))$$

This directory provides a mapping from terms to PeerLists and can be used to identify the peer that maintains the list of Posts for a term t within the directory. We consider a query q as a set of $(term, weight)$-pairs and the set of possible queries as $Q := 2^{T \times \mathbb{R}}$. In order to process a query q, a set of promising peers for this query has to be determined in a database selection step using a function

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

$$selection(q) := comb(\bigcup_{(t,w) \in q} systerms(t), q)$$

that selects a subset of peers by appropriately combining the results from $systerms$ using a function $comb : 2^{P \times S} \times Q \rightarrow 2^P$.

The execution of a query q is a function $exec : Q \times 2^P \rightarrow 2^D$ that sends the query to the peers previously determined by $selection(q)$ and combines the peers' local results into one single final result set (result merging). Finally, we can define the global query execution function $result : Q \rightarrow 2^D$ that is evaluated as

$$result(q) := exec(q, selection(q))$$

7 Implementation

Figure 5 illustrates the architecture of a *MINERVA P2P Search* peer. The peer is layered on top of the distributed hash table (DHT) that builds the global directory by providing mappings from terms to peers. The directory returns 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.

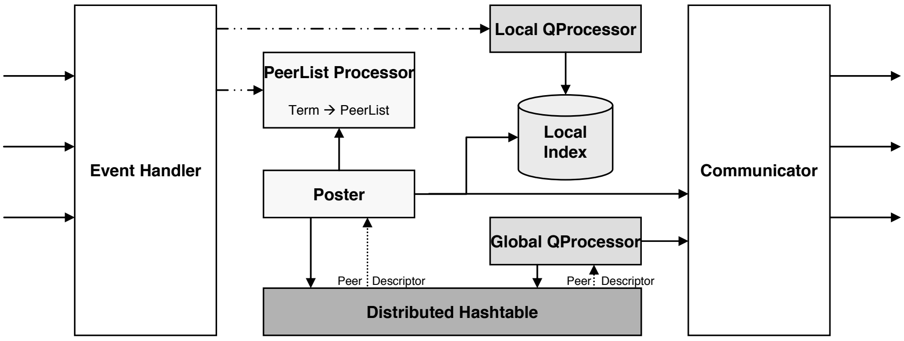
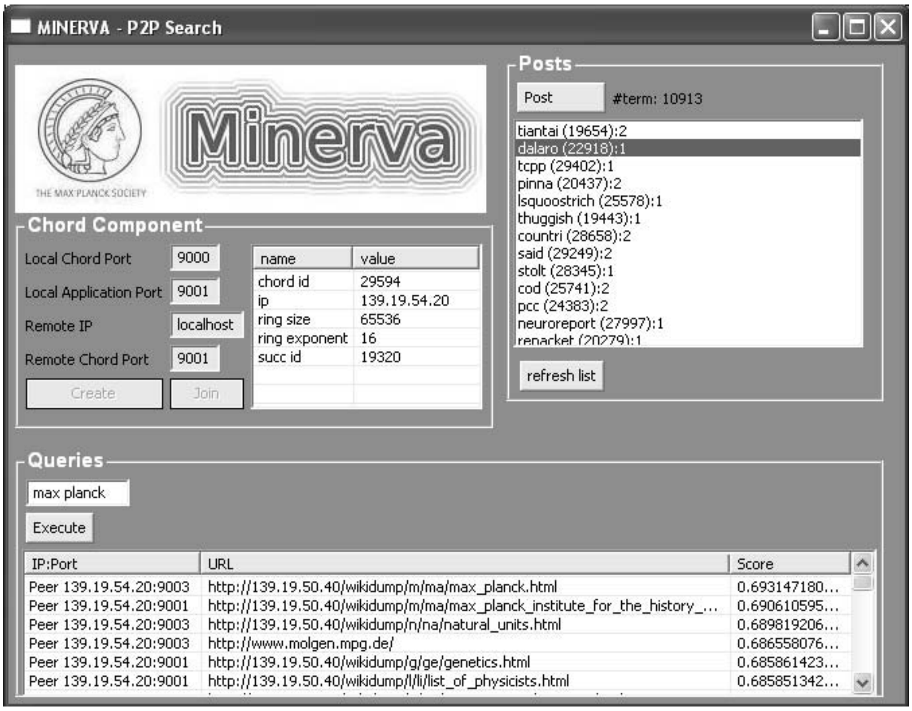


Figure 5: System Architecture

Every peer has a local index. 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 responsible peer; 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 responsible peer and retrieves the respective PeerLists from the PeerList Processors using Communicator components. After running peer selection strategies on these lists, the Global QueryProcessor forwards the complete query to the 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.



Local Chord Port: 9000

Local Application Port: 9001

Remote IP: localhost

Remote Chord Port: 9001

name	value
chord id	29594
ip	139.19.54.20
ring size	65536
ring exponent	16
succ id	19320

#term: 10913

tiantai (19654):2
dalaro (22918):1
tcpp (29402):1
pinna (20437):2
lsquoostrich (25578):1
thuggish (19443):1
countri (28658):2
said (29249):2
stolt (28345):1
cod (25741):2
pcc (24383):2
neuroreport (27997):1
renacket (20279):1

max planck

IP:Port	URL	Score
Peer 139.19.54.20:9003	http://139.19.50.40/wikidump/m/ma/max_planck.html	0.693147180...
Peer 139.19.54.20:9001	http://139.19.50.40/wikidump/m/ma/max_planck_institute_for_the_history_...	0.690610595...
Peer 139.19.54.20:9003	http://139.19.50.40/wikidump/n/na/natural_units.html	0.689819206...
Peer 139.19.54.20:9003	http://www.molgen.mpg.de/	0.686558076...
Peer 139.19.54.20:9001	http://139.19.50.40/wikidump/g/ge/genetics.html	0.685861423...
Peer 139.19.54.20:9001	http://139.19.50.40/wikidump/l/li/list_of_physicists.html	0.685851342...

Figure 6: Prototype GUI

MINERVA uses a Java-based reimplementation of Chord [SMK$^+$01] 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 [ACK04] peers can easily be included to support an arbitrarily heterogeneous environment. Figure 6 shows a screenshot of MINERVA's user interface. The user starts 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. Joining an existing system requires additional information about one existing peer. Status information regarding the Chord ring is displayed and continuously updated. The Posts section provides information about the terms that the 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 directory. The Queries section executes queries with multiple keywords entered into a form field. The results are displayed ordered by their scores.

8 Peer Selection Strategies

Peer selection is the problem of identifying peers that can answer the query of a given peer p_0 with high result quality at low execution costs. Our approach consists of two steps:

1. *Look up* potential candidates
2. *Identify* the best k peers (where k is a system-configuration parameter) by assessing all candidates using a $benefit/cost$ ratio

The first step is conducted by retrieving all Posts for each query term from the global directory. For efficiency, we can also limit the directory lookup to return only the best peers for a keyword, based on a score measure (e.g., number of matches above some score threshold). In the second step, all Posts of a peer are combined to assess the expected result quality (benefit) of the peer for this query, i.e., to perform peer selection. In this step the benefit estimation refers to a peer's contribution to better search results, and the cost estimation refers to the resource consumption and responsiveness of the peers that are involved in a query and also the network performance. For dynamic cost estimation, many proposals exist in the literature on distributed load sharing (see, e.g., [Lu93]). A very simple approach is to disregard the different performance capacities and utilizations of the peers and network routes and merely assume that the overall execution cost of a query increases proportionally with the number of peers that participate in the query. In this paper we adopt this simple assumption and concentrate on the issue of benefit estimation.

The following sections discuss various benefit estimators for peer selection strategies; many other approaches have been proposed in the distributed IR literature, one of the most prominent is the decision-theoretic framework by [Fu99]. Following the terminology of the existing literature, we refer to various statistical measures as *per collection* measures; in our P2P context a collection is the local index content of a peer. We consider only queries with equally weighted terms; so a query is simply a set of terms.

8.1 $cdf - ctf^{max}$ Approach

This very simple heuristic approach combines the *collection document frequency* (cdf) with the *maximum collection term frequency* (ctf^{max}) (the maximum number of term occurrences in the documents of the collection) and summarizes over all query terms. Here the cdf for term t of peer p_i is the number of documents containing t that p_i holds in its local index, and the ctf^{max} value for term t of peer p_i is the term frequency (tf) of t in the document d at p_i that has the highest tf of t among all documents at p_i.

The collection score s_i of the i-th peer with regard to a query q is computed as

$$s_i = \sum_{t \in Q} \alpha \cdot \log cdf_{i,t} + (1 - \alpha) \cdot \log ctf_{i,t}^{max}$$

The value of the parameter α can be chosen between 0 and 1 and is used to emphasize the importance of cdf vs. ctf^{max}. The scores s_i are computed for all peers and sorted in descending order to obtain the final peer ranking.

8.2 CORI-like Approaches

In this section we consider two approaches corresponding to the strategies presented in [CLC95, Ca00]. We refer to these strategies as $CORI$ 1 and $CORI$ 2, respectively.

8.2.1 CORI 1

This approach computes the collection score s_i of the i-th peer with regard to a query q in the following manner:

$$s_i = \sum_{t \in q} \frac{s_{i,t}}{|q|}$$

$$s_{i,t} = \alpha + (1 - \alpha) \cdot T_{i,t} \cdot I_{i,t}$$

The computations of $T_{i,t}$ and $I_{i,t}$ use the size of the underlying Chord identifier space as an upper bound for the *number of peers* in the system, denoted np, the *collection document frequency* (*cdf*), and the *maximum collection document frequency* (cdf^{max}):

$$T_{i,t} = \beta + (1 - \beta) \cdot \frac{log(cdf_{i,t} + 0.5)}{log(cdf_{i,t}^{max} + 1.0)}$$

$$I_{i,t} = \frac{\frac{log(np+0.5)}{cf_t}}{log(np + 1)}$$

where the *collection frequency* cf_t is the number of peers that contain the term t. We approximate this value by the number of peers that have published Posts for term t, i.e., the length of the PeerList for t. The values α and β are chosen as $\alpha = \beta = 0.4$ [CLC95].

8.2.2 CORI 2

This approach was proposed in [Ca00] and differs in the computation of $T_{i,t}$. It considers the size V_i of the term space of a peer (i.e., the total number of distinct terms that the peer holds in its local index) and the average term space size V^{avg} over all peers that contain term t:

$$T_{i,t} = \frac{cdf_{i,t}}{cdf_{i,t} + 50 + 150 \cdot \frac{|V_i|}{|V^{avg}|}}$$

In practice, it is difficult to compute the average term space size over all *peers in the system* (regardless of whether they contain query term t or not). We approximate this value by the average over all *collections found in the PeerLists*.

8.3 GlOSS-like Approach

This strategy is based on the work presented in [GGMT99]; we refer to this strategy as GlOSS-like. First, we need to sort the query terms $t_i \in Q$ in ascending order of their *cdf* values (for simplicity, we re-index the terms $t_1, t_2, ..., t_q$ to reflect this sorting), i.e., it holds for any pair t_t, t_{t+1} that $cdf_t \leq cdf_{t+1}$.

In a second step, the average term frequency per document in a collection (ctf_t^{avg}) is computed by combining the *collection term frequency* (ctf_t - number of occurrences of a term in the entire collection) and the *collection document frequency* (cdf_t):

$$ctf_t^{avg} = \frac{ctf_t}{cdf_t}$$

Now we can calculate the final collection score of the i-th peer with regard to a query q as

$$s_i = \sum_{t=1}^{q} \left[(cdf_{i,t} - cdf_{i,t-1}) \cdot \sum_{u=t}^{q} (ctf_{i,u}^{avg} \cdot log(\frac{|C_i|}{cdf_{i,u}}) \right]$$

where $cdf_{i,0} := 0$ and $|C_i|$ is the number of documents in the i-th collection.

8.4 Language-Model (LM) based Approaches

The next two strategies use statistical language models (LMs) for computing collection rankings.

8.4.1 LM of Callan

This approach builds upon the ideas presented in [SJCO02] to calculate a collection score as

$$s_i = \sum_{t \in Q} log(\lambda \cdot s_{i,t} + (1 - \lambda) \cdot s_{GE,t})$$

where $s_{GE,t}$ is a statistical model of *General English* and λ is a calibration parameter. We approximate the model by calculating the number of occurrences of term t in the collections divided by the overall number of terms in the collections. We choose $\lambda = 0.7$ and compute $s_{i,t}$ as

$$s_{i,t} = \frac{ctf_{i,t}}{cs_i}$$

where cs_i is the size of the i-th peer counted in term occurrences (not distinct terms).

8.4.2 LM of Xu & Croft

The second language-model approach is based on [XC99]. We compute the distance between the query model and the collection model as

$$dist_i = \sum_{t \in Q} \left[\frac{1}{|q|} \cdot log(\frac{1}{|q| \cdot s_{i,t}}) \right] \; with \; s_{i,t} = \frac{ctf_{i,t} + 0.01}{c_{s,i} + 0.01 \cdot |V_i|}$$

The peer ranking (reflecting similarity) corresponds to the ascending order of distances.

9 Experiments

9.1 Experimental Setup

Experiments are conducted on collections that have been created by Web crawls originating from manually selected crawl seeds on the topics Sports, Computer Science, Entertainment, and Life, leading to 10 thematically focused collections. Additionally, one reference collection was created by combining all collections and eliminating duplicates. Table 1 gives details about the collections. Note the overlap between the 10 original collections.

For the query workload we took the 7 most popular queries on AltaVista, as reported by `http://www.wordtracker.com` for September 21, 2004, and 3 additional queries that were specifically suitable for our corpus. Table 2 lists all queries.

For each query we obtain an *ideal peer ranking* as follows: the query is executed on the reference collection with the measures introduced in Section 4 to obtain a reference query

Collection	# Documents	Collection Size in MB
Computer Science	10459	137
Life	12400	144
Entertainment	11878	134
Sport	12536	190
Computer Science mixed	11493	223
Computer Science mixed	13703	239
CS Google	7453	695
Sport Google	33856	1,086
Life Google	16874	809
Entertainment Google	18301	687
Σ	168953	4,348
Combined Collection	142206	3,808

Table 1: Collection Statistics

Max Planck Light Wave Particle*	Einstein Relativity Theory*
Lauren Bacall	Nasa Genesis
Hainan Island	Carmen Electra
National Weather Service	Search Engines
John Kerry	George Bush Iraq*

Table 2: List of used queries (* denotes queries *not* taken from WordTracker)

result. Subsequently, the query is executed on each of the collections individually, using the same strategy. These local results are compared to the reference query result using the *rank distance* function described in detail in Section 9.2. We order the peers in ascending order of these distances to obtain the ideal peer ranking.

We evaluate our peer selection strategies by comparing their peer selection results (peer rankings) to this ideal peer ranking, again using our rank distance function. Figure 7 illustrates this experimental setup.

We have a number of system parameters that influence the experimental results:

- Number of peers returned by peer selection strategies
- Number of documents retrieved from each peer
- Number of documents retrieved from combined collection (ideal document ranking)
- Number of peers in ideal peer ranking

All experiments have been conducted using 10 Peers running as separate processes on a single notebook with a Pentium M 1.6GHz processor and 1GB main memory. All peers share a common Oracle $10g$ database that is installed on a Dual-Pentium Xeon 3GHz processor with 4GB main memory. The peers are connected to the database through a 100MBit network.

9.2 Rank Distance Function

Formally, a ranking σ is a bijection (i.e., a permutation) from a domain D_σ to $[k]$ where $|D_\sigma| = k$ and $[k] := \{1, ..., k\}$. Metrics comparing permutations have intensively been studied for a long time [KG90, DG88]. One of the most prominent metrics is *Spearman's*

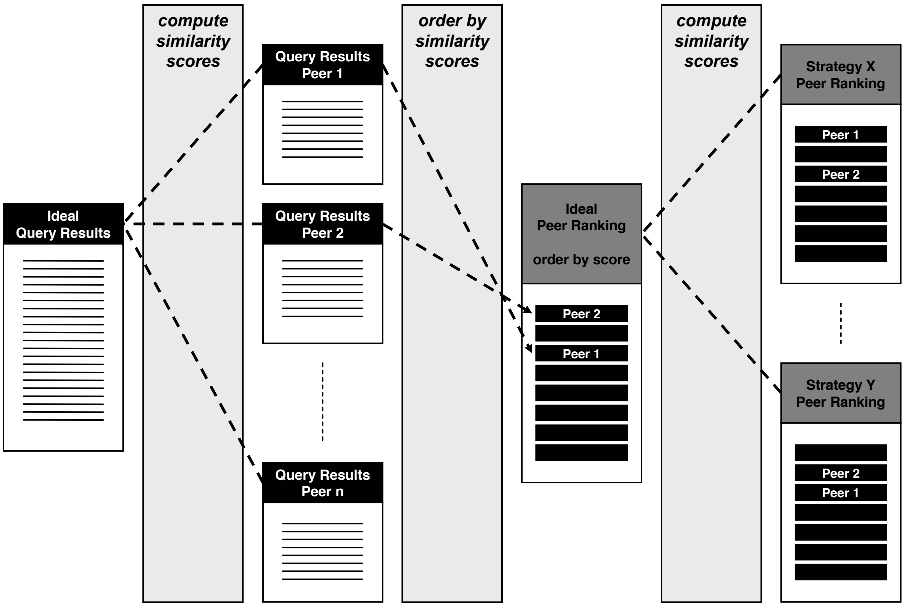


Figure 7: Experimental Setup

footrule metric [DG77, DG88] that calculates the difference between two permutations σ_1 and σ_2 with $D = D_{\sigma_1} = D_{\sigma_2}$ as follows:

$$F(\sigma_1, \sigma_2) := \sum_{i \in D} |\sigma_2(i) - \sigma_1(i)| \tag{1}$$

In our case, the domains D_{σ_1} and D_{σ_2} are not necessarily identical, because the peer selection strategies might not return *all* peers. Thus, we can neither apply Spearman's footrule metric (1) nor other popular techniques like *Kendall's tau* [KG90].

[FKS03] gives an overview about metrics on comparing top-k lists representing 'incomplete' rankings by presenting modifications of the well-known metrics for permutations. One possibility to apply metrics for permutations on top-k lists is to extend the top-k lists to complete rankings over a shared domain, that is, the union of the domains of the top-k lists.

We propose the following formula to calculate the distance between two rankings σ_1 and σ_2:

$$F'(\sigma_1, \sigma_2) := \sum_{i \in D_{\sigma_2}} |\sigma_2(i) - \sigma_1(i)| \tag{2}$$

Although this formula closely resembles Spearman's footrule metric, it sums only over elements that are contained in D_{σ_2}. However, (2) is only valid if D_{σ_1} is a superset of D_{σ_2} because $\sigma_1(i)$ is undefined for all $i \in D_{\sigma_2} \setminus D_{\sigma_1}$. Thus, we need to define an extension ${\sigma_1}'$ of σ_1 as follows:

$$\sigma_1{}'(i) := \begin{cases} \sigma_1(i) & i{\in}D_{\sigma_1} \\ |D_{\sigma_1}|{+}1 & i{\notin}D_{\sigma_1} \end{cases}$$

This extension of σ_1 adds a particularly high penalty for misplaced entries among the top ranks of σ_2. Note that we do not need an extension of σ_2, because we sum only over elements in D_{σ_2}. This causes F' to be asymmetric.

Another important issue appears when trying to evaluate several rankings of different sizes to one reference ranking. To avoid unfair comparison of short result lists with good matches against long result lists with not so good matches at low ranks, we required a minimum size for each query result list from each peer and supplemented shorter lists by "dummy" documents that would correspond to the rank $|D_{\sigma_1}| + 1$ in the reference collection.

9.3 Experimental Results

Figure 8 (a) shows the distances between the peer rankings returned by the peer selection strategies and the ideal peer ranking. The distances for each strategy are averaged over the 10 queries in our benchmark, based on the rank distance definition given in Section 9.2. For these test quries, CORI2 produced the best results and clearly outperformed the GLOSS-based strategy and both Language Modeling approaches. The very simple $cdf - ctf^{max}$ approach worked remarkably well provided that α was not set too low. The poor performance of the Language Modeling approaches contradicts the findings by [SJCO02], but this effect may be heavily dependent on the specific nature of the queries and the distributed corpora (e.g., the degree to which corpora of different peers may overlap).

Using the same set of collections and queries, we studied the recall relative to the top-30 documents from the combined collection when we query the peers according to their positions in the ideal peer ranking. Figure 8 (b) shows that sending the query to the best peer only already yielded an average of about 60% of all relevant documents, whereas the inferior peers typically do not contribute any new documents to the query result. Note that this does not mean they do not *contain* any relevant documents, but rather that their relevant documents have already been contributed by other peers before. So taking into account the potential overlap of the peers' local contents is crucial.

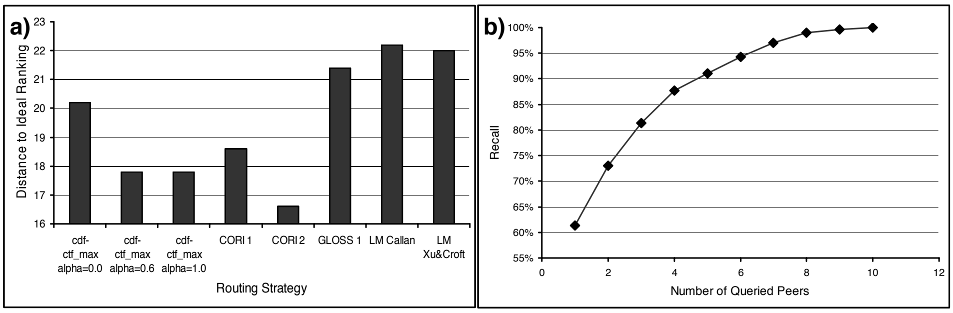


Figure 8: Experimental Results

The importance of dealing with redundancy caused by overlapping indexes of different peers becomes even clearer when looking at the query execution time and the number of relevant documents obtained, as the number of queried peers increases. Figure 9 (c) shows that, in our experiments conducted on one single computer, the query execution time increases linearly; in a real-world scenario, the execution time is expected to remain nearly constant as the load is spread over a number of independent processors. Note that, given the high computational load during the experiments, a query is executed in a reasonable time of 2 seconds per peer. This time is dominated by the execution time at the peers; the system overhead is almost negligible. Figure 9 (b) shows the number of relevant documents obtained when increasing the number of peers absolute and additional. As expected, less relevant peers add only a very small number of relevant documents. The ratio between the number of relevant documents and the execution time (Figure 9(a)) demonstrates that, in our experimental setup, the best ratio is already achieved when asking only 1 remote peer (in addition to the local evaluation at the querying peer itself).

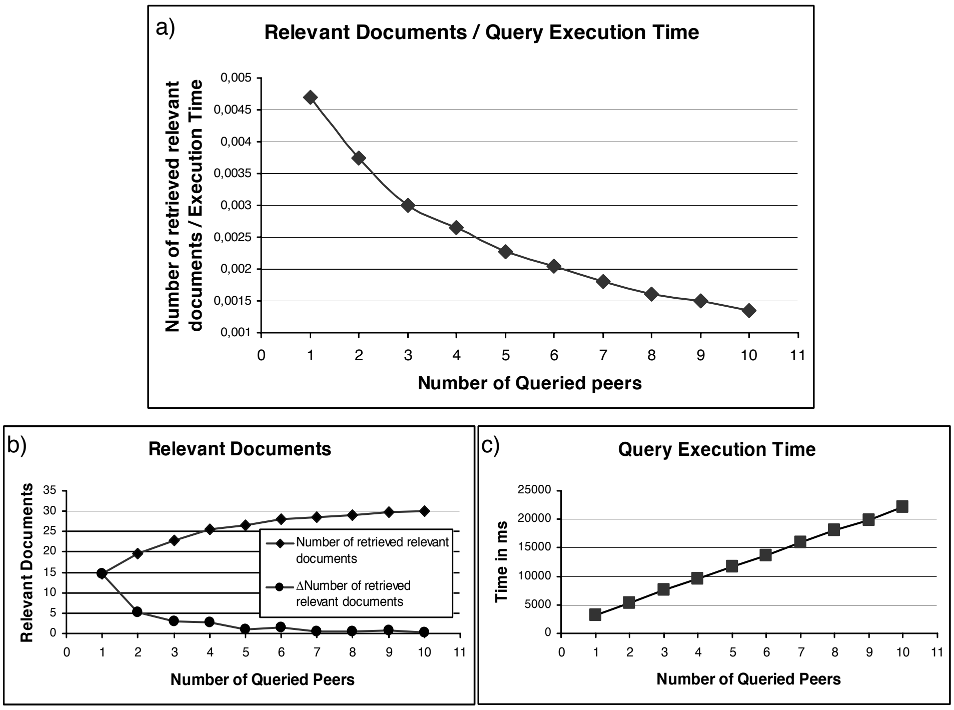


Figure 9: Performance Evaluation

We also studied the run-time overhead and efficiency of our techniques. In our experiments we observed the following performance characterisics (which are independent of the peer selection strategies). The size of a single Post is about 10 to 20 bytes for typical keywords. The overall amount of data that has to be transfered over the network during a complete posting process for a collection containing $45\ 000$ terms is about 650KB. Note that we used a compression technique to reduce the size of the messages. A single PeerList request and the query itself only account for 100-200 bytes for a typical two-keyword query. The complexity of query routing strategies that obtain the PeerLists from the preceding step is $O(nl + mlog(m))$ where n is the number of query terms and l is the maximum PeerList length. nl is an upper bound for the number of Posts that have to be processed before the peer ranking (at most m entries) has to be sorted in $O(mlog(m))$, where m is the number of distinct peers found in the PeerLists.

10 Conclusion and Future Work

This paper has presented the MINERVA project on building a P2P Web search engine. We described its novel architecture, revisited existing approaches to the database selection problem in distributed IR, and adapted these strategies to fit our system environment. Our preliminary experiments show significant differences among the peer selection strategies in terms of peer rankings and resulting search result quality. In our setting, CORI-like approaches performed best and even a very simple frequency-based heuristics outperformed the more sophisticated approaches based on conceptually richer statistical language models. However, these results may be dependent on the specific nature of queries and corpora, in particular, the degree to which the local index contents of different peers may overlap. Our early studies also showed that the run-time overhead of MINERVA is fairly low and that for most queries sending the query to only one remote peer achieved the best quality/cost ratio.

We are currently preparing experiments on a larger document collection with a much larger number of peers and a broader variety of queries including queries with many keywords generated by automatic query expansion techniques. We are also working on enhancing the peer selection strategies themselves by incorporating bookmark statistics and addressing the overlap problem among peers [BMWZ04]. Ideally, a query that has already been executed locally on the querying peer should be sent only to peers that are likely to provide complementary additional results (i.e., that show only little overlap with the query initiator). If a remote peer only yields the high-quality results that the query initiator already knows from its local evaluation, the remote peer is useless.

References

[ACK04] Alonso, G., Casati, F., and Kuno, H.: *Web Services - Concepts, Architectures and Applications*. Springer. Berlin;Heidelberg;New York. 2004.

[ACMHP04] Aberer, K., Cudre-Mauroux, P., Hauswirth, M., and Pelt, T. V.: Gridvine: Building internet-scale semantic overlay networks. Technical report. EPFL. 2004.

[APHS02] Aberer, K., Punceva, M., Hauswirth, M., and Schmidt, R.: Improving data access in p2p systems. *IEEE Internet Computing*. 6(1):58–67. 2002.

[BB04] Buchmann, E. and Böhm, K.: How to Run Experiments with Large Peer-to-Peer Data Structures. In: *Proceedings of the 18th International Parallel and Distributed Processing Symposium, Santa Fe, USA*. April 2004.

[BMWZ04] Bender, M., Michel, S., Weikum, G., and Zimmer, C.: Bookmark-driven query routing in peer-to-peer web search. In: Callan, J., Fuhr, N., and Nejdl, W. (Hrsg.), *Proceedings of the SIGIR Workshop on Peer-to-Peer Information Retrieval*. S. 46–57. 2004.

[Ca00] Callan, J.: Distributed information retrieval. *Advances in information retrieval, Kluwer Academic Publishers*. S. 127–150. 2000.

[CAPMN02] Cuenca-Acuna, F. M., Peery, C., Martin, R. P., and Nguyen, T. D.: PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. Technical Report DCS-TR-487. Rutgers University. September 2002.

[CFK03] Cohen, E., Fiat, A., and Kaplan, H.: Associative search in peer to peer networks: Harnessing latent semantics. In: *Proceedings of the IEEE INFOCOM'03 Conference, April 2003*. April 2003.

[CGM02a] Crespo, A. and Garcia-Molina, H.: Routing indices for peer-to-peer systems. In: *Proc. of the 28th Conference on Distributed Computing Systems*. July 2002.

[CGM02b] Crespo, A. and Garcia-Molina, H.: Semantic Overlay Networks for P2P Systems. Technical report. Stanford University. October 2002.

[Ch02] Chakrabarti, S.: *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann. San Francisco. 2002.

[CL03] Croft, W. B. and Lafferty, J.: *Language Modeling for Information Retrieval*. volume 13. Kluwer International Series on Information Retrieval. 2003.

[CLC95] Callan, J. P., Lu, Z., and Croft, W. B.: Searching distributed collections with inference networks. In: *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*. S. 21–28. ACM Press. 1995.

[DG77] Diaconis, P. and Graham, R.: Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society*. S. 262–268. 1977.

[DG88] Diaconis, P. and Graham, R.: Group representation in probability and statistics. *Institute of Mathematical Statistics*. 1988.

[Fa99] Fagin, R.: Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.* 58(1):83–99. 1999.

[FKS03] Fagin, R., Kumar, R., and Sivakumar, D.: Comparing top k lists. In: *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*. S. 28–36. Society for Industrial and Applied Mathematics. 2003.

[FLN01] Fagin, R., Lotem, A., and Naor, M.: Optimal aggregation algorithms for middleware. In: *Symposium on Principles of Database Systems*. 2001.

[Fu99] Fuhr, N.: A decision-theoretic approach to database selection in networked IR. *ACM Transactions on Information Systems*. 17(3):229–249. 1999.

[GBK00] Guntzer, U., Balke, W.-T., and Kiesling, W.: Optimizing multi-feature queries for image databases. In: *The VLDB Journal*. S. 419–428. 2000.

[GBS01] Grabs, T., Böhm, K., and Schek, H.-J.: Powerdb-ir: information retrieval on top of a database cluster. In: *Proceedings of the tenth international conference on Information and knowledge management*. S. 411–418. ACM Press. 2001.

[GGMT99] Gravano, L., Garcia-Molina, H., and Tomasic, A.: Gloss: text-source discovery over the internet. *ACM Trans. Database Syst.* 24(2):229–264. 1999.

[KG90] Kendall, M. and Gibbons, J. D.: Rank correlation methods. *Edward Arnold, London.* 1990.

[KLL$^+$97] Karger, D., Lehman, E., Leighton, T., Levine, M., Lewin, D., and Panigrahy, R.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: *ACM Symposium on Theory of Computing*. S. 654–663. May 1997.

[LC03] Lu, J. and Callan, J.: Content-based retrieval in hybrid peer-to-peer networks. In: *Proceedings of the twelfth international conference on Information and knowledge management*. S. 199–206. ACM Press. 2003.

[LNS$^+$03] Löser, A., Naumann, F., Siberski, W., Nejdl, W., and Thaden, U.: 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)*. S. 33–47. 2003.

[Lu93] Ludwig, T. Lastverwaltung für parallelrechner. 1993.

[MRYGM01] Melnik, S., Raghavan, S., Yang, B., and Garcia-Molina, H.: Building a distributed full-text index for the web. *ACM Trans. Inf. Syst.* 19(3):217–241. 2001.

[MS99] Manning, C. D. and Schütze, H.: *Foundations of Statistical Natural Language Processing*. The MIT Press. Cambridge, Massachusetts. 1999.

[MYL02] Meng, W., Yu, C. T., and Liu, K.-L.: Building efficient and effective metasearch engines. *ACM Computing Surveys*. 34(1):48–89. 2002.

[NF03] Nottelmann, H. and Fuhr, N.: 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*. S. 290–297. ACM Press. 2003.

[NR99] Nepal, S. and Ramakrishna, M. V.: Query processing issues in image (multimedia) databases. In: *ICDE*. S. 22–29. 1999.

[RD01] Rowstron, A. and Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*. S. 329–350. 2001.

[RFH$^+$01] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S.: A scalable content-addressable network. In: *Proceedings of ACM SIGCOMM 2001*. S. 161–172. ACM Press. 2001.

[RV03] Reynolds, P. and Vahdat, A.: Efficient peer-to-peer keyword searching. In: *Proceedings of International Middleware Conference*. S. 21–40. June 2003.

[SJCO02] Si, L., Jin, R., Callan, J., and Ogilvie, P.: A language modeling framework for resource selection and results merging. In: *Proceedings of the eleventh international conference on Information and knowledge management*. S. 391–397. ACM Press. 2002.

[SMK$^+$01] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H.: 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*. S. 149–160. ACM Press. 2001.

[SMW$^+$03] Suel, T., Mathur, C., Wu, J., Zhang, J., Delis, A., Kharrazi, M., Long, X., and Shanmugasunderam, K.: Odissea: A peer-to-peer architecture for scalable web search and information retrieval. Technical report. Polytechnic Univ. 2003.

[TWS04] Theobald, M., Weikum, G., and Schenkel, R.: Top-k query evaluation with probabilistic guarantees. *VLDB*. S. 648–659. 2004.

[TXD03] Tang, C., Xu, Z., and Dwarkadas, S.: 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*. S. 175–186. ACM Press. 2003.

[WGdW03] Wang, Y., Galanis, L., and de Witt, D. J.: Galanx: An efficient peer-to-peer search engine system. *Available at http://www.cs.wisc.edu/ yuanwang*. 2003.

[WMYL01] Wu, Z., Meng, W., Yu, C. T., and Li, Z.: Towards a highly-scalable and effective metasearch engine. In: *World Wide Web*. S. 386–395. 2001.

[XC99] Xu, J. and Croft, W. B.: Cluster-based language models for distributed retrieval. In: *Research and Development in Information Retrieval*. S. 254–261. 1999.

[YGM02] Yang, B. and Garcia-Molina, H.: Improving search in peer-to-peer networks. In: *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*. S. 5–14. IEEE Computer Society. 2002.