Improving Collection Selection with Overlap Awareness in P2P Search Engines*

Matthias Bender†, Sebastian Michel†, Peter Triantafillou‡,
Gerhard Weikum†, Christian Zimmer†
† Max-Planck-Institut für Informatik
66123 Saarbrücken, Germany
‡University of Patras
Rio, 26500, Greece
† {mbender, smichel, weikum, czimmer}@mpi-sb.mpg.de
‡ peter@ceid.upatras.gr

ABSTRACT

Collection selection has been a research issue for years. Typically, in related work, precomputed statistics are employed in order to estimate the expected result quality of each collection, and subsequently the collections are ranked accordingly. Our thesis is that this simple approach is insufficient for several applications in which the collections typically overlap. This is the case, for example, for the collections built by autonomous peers crawling the web. We argue for the extension of existing quality measures using estimators of mutual overlap among collections and present experiments in which this combination outperforms CORI, a popular approach based on quality estimation. We outline our prototype implementation of a P2P web search engine, coined MINERVA[1], that allows handling large amounts of data in a distributed and self-organizing manner. We conduct experiments which show that taking overlap into account during collection selection can drastically decrease the number of collections that have to be contacted in order to reach a satisfactory level of recall, which is a great step toward the feasibility of distributed web search.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*selection process, information filtering*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed Systems*

General Terms

Algorithms, Design, Experimentation

Keywords

Peer-to-Peer information systems, distributed IR, query routing, overlap estimation

1. INTRODUCTION

1.1 Motivation

Collection selection has been a popular research area in distributed information retrieval for many years. Most of the related work on this issue use pre-computed statistical information to estimate the expected result quality of different collections en route to identifying the most promising sources for an information need. A key goal from a performance viewpoint is to minimize the number of individual collections that have to be gathered in order to achieve good result quality (usually measured in terms of recall in this distributed setting).

Also in recent years, the Peer-to-Peer (P2P) paradigm has received increasing attention. While becoming popular mainly in the context of filesharing systems such as Gnutella or KaZaA, P2P has found its way into distributed information retrieval due to its ability to handle huge amounts of data in a distributed and self-organizing way. In a typical P2P 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.

As such, research in P2P searching enjoys a large overlap with research on distributed information retrieval and can highly benefit from existing work. However, the peculiarities of such an architecture require a different view on some key aspects. For example, the absence of a centralized indexing facility together with the difficulties to calcu-

*Partially supported by the EU within the 6th Framework Programme under contract 001907 "Dynamically Evolving, Large Scale Information Systems" (DELIS).
[1]http://www.minerva-project.org

SIGIR'05, August 15–19, 2005, Salvador, Brazil.

late global metrics in this large and highly dynamic network hamper the use of traditional methods for collection selection. Also, the absence of a central control instance causes the peers to learn about portions of the web in a largely uncoordinated manner. Given typical popularity distributions on today's web, for example, it becomes obvious that these autonomous peers will not form disjoint partitions of their combined document space, but rather a highly overlapping set of collections.

Existing collection selection approaches taking into account only the expected result quality of a collection will inevitably lose some of their power in a P2P setting. For example, consider a scenario where two peers with high interest in current affairs have (independently of each other) crawled large fractions of a popular news site, such as cnn.com. A traditional approach to collection selection is likely to rank both peers high given a related query, even though their results may highly overlap. Because the second peer is likely to not add many new documents to the query result, the performance of the query can be improved if another, complementary peer is chosen instead.

1.2 Contribution

Our main contribution lies in showcasing the dramatic performance improvement possible by employing estimators of mutual overlap of collections. We will also present a novel technique for estimating a query-specific collection overlap and a novel way to combine a popular quality estimation metric with overlap estimators. Our overall goal is to make large-scale distributed search feasible. With this paper, we hope to make a decisive step toward this goal. We have implemented our new technique within our prototype P2P web search engine and we quantify the performance improvement of our contribution.

Section 2 gives an overview of related research in the different fields that we touch with our work. Section 3 presents the architecture of a distributed P2P search engine that was used for our experiments. Section 4 briefly introduces CORI, a popular approach for collection selection, that is used as a baseline reference point for our work. Section 5 discusses one approach to quantify overlap and novelty into the collection selection process and introduces the necessary set of tools, in particular Bloom filters. Section 6 describes how we combine the measures for quality and novelty in a two-step approach. Section 7 presents a number of experiments to show the benefits of our approach. Section 8 concludes and briefly discusses future research directions.

2. RELATED WORK

In recent years, many approaches have been proposed for collection selection in distributed IR, among the most prominent the decision-theoretic framework by [11], the GlOSS method presented in [14], and approaches based on statistical language models [22, 27]. [5] gives an overview of algorithms for distributed IR style result merging and database content discovery. [11] presents a formal decision model for database selection in networked IR. [19] investigates different quality measures for database selection. [13, 16] study scalability issues for a distributed term index. None of the presented techniques incorporates overlap detection into the selection process.

Estimating overlap of sets has been receiving increasing attention recently for modern emerging applications, such as data streams, internet content delivery, etc. [4] describes a permutation-based technique for efficiently estimating set similarities for informed content delivery. [12] proposes a hash-based synopsis data structure and algorithms to support low-error and high-confident estimates for general set expressions. Bloom [2] describes a data structure for succinctly representing a set in order to support membership queries. [17] proposes compressed Bloom filters that improve performance in a distributed environment where network bandwidth is an issue.

[10] describes the use of statistics in ranking data sources with respect to a query. They use probabilistic measures to model overlap and coverage of the mediated data sources, but do not mention how to acquire these statistics. In contrast, we assume these statistics being generated by the participating peers (based on their local collections) and present a DHT based infrastructure to make these statistics globally available.

[28] considers novelty and redundancy detection in a centralized, document-stream based information filtering system. Although the technique presented seems to be applicable in a distributed environment for filtering the documents at the querying peer, it is not obvious where to get these documents from. In a large-scale system, it seems impossible to query all peers and to process the documents.

[18, 15] have also worked on overlap statistics in the context of collection selection. They present a technique to estimate coverage and overlap statistics by query classification and data mining and use a probing technique to extract features from the collections. Expecting that data mining techniques will be very heavy for the envisioned, highly-dynamic application environment, we adopt a different philosophy. We adapt statistical methods and use them to estimate the overlap between data collections and we develop novel algorithms that utilize these for efficiently selecting the best collections during query processing.

Recent research on P2P systems, such as Chord [23], CAN [20], Pastry [21], P2P-Net [3], 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 n, 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 incorporate algorithms to cope with failures and the dynamics of a P2P system as peers join or leave the system in an unpredictable manner. However, the approaches are limited to the exact matching of keys, making them suitable for single keyword queries on keys. This is however highly inappropriate when queries should return a ranked result list of the most relevant approximate matches [7].

In the following we briefly discuss some prior and ongoing projects toward P2P Web search.

Galanx [26] is a P2P search engine implemented using the Apache HTTP server and BerkeleyDB. The Web site servers are the peers of this architecture; pages are stored only where they originate from. In contrast, our approach leaves it to the peers to what extent they want to crawl interesting fractions of the Web and build their own local indexes.

PlanetP [8] is a publish-subscribe service for P2P communities, supporting content ranking search. PlanetP dis-

tinguishes 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 appears to be limited to a few thousand peers.

Odissea [24] assumes a two-layered search engine architecture with a global index structure distributed over the nodes in the system. A single node holds the complete, Web-scale, index for a given text term (i.e., keyword or word stem). Query execution uses a distributed version of Fagin's threshold algorithm [9]. The system appears to cause high network traffic when posting document metadata into the network, and the presented query execution method seems limited to queries with at most two keywords. The paper actually advocates using a limited number of nodes, in the spirit of a server farm.

3. THE MINERVA P2P SEARCH ENGINE ARCHITECTURE

We assume a P2P collaboration in which every peer is autonomous and has a local index that can be built from the peer's own crawls or imported from external sources and tailored to the user's thematic interest profile. The index contains inverted lists with URLs for Web pages that contain specific keywords (a.k.a. terms).

A conceptually global but physically distributed directory, which is layered on top of a Chord-style Dynamic Hash Table (DHT), holds compact, aggregated information about the peers' local indexes and only to the extent that the individual peers are willing to disclose. We use the 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, availability, and load balancing, the directory entry for a term may be replicated across multiple peers. Thus, peers in our system can play two roles: as directory peers and as peers storing index lists.

Directory maintenance, peer selection, and query processing work as follows. First, every peer with a local index publishes a summary (*Post*) about every term in its local index to the directory. A hash function is applied to the term in order to determine the directory peer currently responsible for this term. This directory peer maintains a *PeerList* of all postings for this term from peers across the network. Posts contain contact information about the peer who posted this summary together with statistics to calculate IR-style measures for a term (e.g., the size of the inverted list for the term, the maximum average score among the term's inverted list entries, or some other statistical measure). These statistics are used to support the peer selection process, i.e., determining the most promising peers for a particular query.

The querying process for a multi-term query proceeds as follows: first, the query is executed locally using the peer's local index. If the result is considered unsatisfactory by the user, the querying peer retrieves a list of potentially useful peers by issuing a *PeerList request* for each query term to the underlying overlay-network directory. Using database selection methods from distributed IR and metasearch [5], a number of promising peers for the complete query is computed from these PeerLists. This step is referred to as *peer selection*. Subsequently, the query is forwarded to these peers and executed based on 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; this step is referred to as *result merging*.

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 appropriate peer selection methods to limit the number of peers involved in a query keeps the size of the global directory manageable and reduces network traffic, while at the same time allowing the query execution itself to rely on comprehensive local index data. We expect this approach to scale very well as more and more peers jointly maintain the 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, information about relevance assessments (e.g., derived from peer-specific query logs or click streams), or explicit user feedback. This information could be leveraged when executing a query to further enhance result quality.

4. COLLECTION SELECTION

The following section briefly introduces CORI [5], one of the best and most popular benefit estimators for collection selection strategies that tries to estimate the expected result quality of a collection using aggregated statistics about the collections.

Following the terminology of the existing literature [6, 5], 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.

CORI computes the collection score s_i of the i-th peer with regard to a query $Q = \{t_1, t_2, ..., t_n\}$ 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 *number of peers* in the system, denoted np, the *document frequency* (cdf) of term t in collection i, and the *maximum document frequency* (cdf^{max}) for any term t in collection i:

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

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

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 $\alpha = \beta = 0.4$ [6].

CORI 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:

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*.

5. NOVELTY ESTIMATION

As we have motivated in the introduction, considering overlap is a crucial task in order to make large-scale distributed IR feasible. While existing approaches to collection selection typically try to estimate the relevance of a collection to a given query, they do not consider the overlap with other, previously contacted collections. However, it is obviously inefficient to choose peers based on their relevance only, as relevant documents are completely worthless if they have been delivered by other collections before.

In our context, when estimating overlap, what we are actually interested in is the *novelty* of a peer with respect to a given reference collection C_{ref}: specifically, given a representation of the document space that has already been covered (e.g., by a local index or by other peers previously queried), we are interested in the contribution that the collection C_i of peer P_i can add to this space. More formally, we define novelty as $|C_p| - |C_p \cap C_{ref}|$.

In this section, we discuss one approach of estimating the novelty of collections. Doing so in a distributed environment is a non-trivial task that becomes even more complicated as we are not interested in the overall novelty between collections, but rather the novelty that the collections show in their results for a particular query.

In related literature one can find numerous techniques to represent sets in a compact way. For our experiments, we use Bloom filters, one of the most popular technique for this purpose and we develop our novelty estimator using them. We want to emphasize, however, that our design is fundamentally independent of the specific technique used and allows for any other approach as well, as long as it produces numeric scores for collection novelty.

5.1 System Architecture Revisited

We want to piggy-back all information necessary for novelty calculation onto the *Post* messages described in Section 3. This choice avoids having to retrieve this information at query time from a carefully selected subset of all peers which would add an extra network round-trip communication and, thus, increase the latency encountered when executing a query.

In order to be able to estimate the overlap between collections, each collection publishes an appropriate summary about the documents it contains. Note that these summaries are fundamentally different from the summaries introduced before that are used to estimate the result quality of peers. Obviously, there are at least two conceptually different ways to create and publish these summaries:

a) Create a summary for the complete collection and publish it on a per-peer basis using a separate DHT-based directory.

b) Create term-specific summaries for (potentially a subset of) all terms and include them with the standard post process.

While the first strategy might seem to be more efficient in terms of storage and bandwidth usage, we argue that strategy b) is preferable for two reasons: First, for query performance reasons, since it does not require an additional directory lookup for each peer and, second, since it facilitates quality and novelty estimation of the peers' collections specifically for the queried terms. It is instructing to note that a combination of these strategies in which each peer would include a summary in the style of strategy a) with every per-term post as illustrated in strategy b) would introduce an enormous degree of redundancy. This would go against the idea of efficiently using storage and bandwidth. At the same time, it would suffer from the lack of query-specificity illustrated before.

We assume that each peer holds one Bloom filter summarizing each of its index lists; these can easily be precomputed and stored in a database. We expect index lists (and, thus, their Bloom filters) to be rather static, so that summaries need not be reconstructed frequently. During the regular Post process, these per-term Bloom filters are added to the posts. Note that Bloom filters like all sparse bit vectors can be compressed very well using standard compression techniques like gzip in order to save bandwidth and storage resources. Also, there are enhancements that produce compressed Bloom filters [17] in order to save bandwidth and storage resources.

5.2 Bloom Filters

A Bloom filter (BF) [2] is a simple data structure that represents a set as a bit vector in order to efficiently (in time and space) support membership queries. With bit vectors being a very compact representation of a set, Bloom filters are an ideal representation in our environment where storage and bandwidth consumption is an issue.

For a particular set, a Bloom filter is a bit map of length m and is created by applying k hash functions on each member document, each yielding a bit location in the vector. Exactly (and only) these positions of the Bloom filter will be set to 1. To check if a given element is in the set, the element is hashed using the same hash function and the corresponding k bits of the Bloom filter are examined. If there is at least one of these bits that is *not* set to 1, the element is definitely not in the set; otherwise it is conjectured that it is in the set: There is a non-zero probability that the examined k bit positions were set by other documents, thus, creating a *false positive*. The probability of a false positive can be calculated by $pfp \approx (1 - e^{-kn/m})^k$ where n is the number of items in the original set.

5.3 Handling Multi-Keyword Queries

Given the Bloom filter summaries for individual index lists, we need a way to combine these in order to estimate the novelty of collections with respect to multi-keyword queries. The querying peer receives for each query term a number of posts that contain summaries of the peers' content plus a Bloom filter that represents the index list for a particular term at a particular peer. Now we consider a Peer P_i and the set of its posted Bloom filters $\{bf_{i,t_1}, bf_{i,t_2}, bf_{i,t_2}, ..., bf_{i,t_r}\}$ for the query terms $\{t_1, t_2, t_3, ..., t_r\}$. In order to get a summary that represents the set of documents of peer P_i that contain all r terms (i.e., that are likely to be in the result list that this peer would return), we combine the peer's r Bloom filters as follows:

Given two Bloom filters with the same hash function and of equal length, it is possible to calculate the Bloom filter that represents the intersection of the two sets by simply combining both filters (i.e. bit vectors) using a bitwise *AND* operation (cf. Figure 1). We denote this operation by &. For a set $\{bf_1, bf_2, bf_3, ..., bf_r\}$ of Bloom filters, the combined Bloom filter is given by $bf_i := bf_1 \& bf_2 \& bf_3 \& ... \& bf_r$. This approximates the coverage of each Peer P_i with regard to the query.

Figure 1: Combining Bloom Filters for Multi-Keyword Queries

This calculation iterates for every query term.

5.4 Estimating Novelty

Given Bloom filter representations of the already seen document space and of the peers in question, we need to estimate the expected contribution (*novelty*) of each peer P_i to the query result. For this purpose, we compare P_i's Bloom filter bf_i (that was obtained by combining its individual Bloom filters as described in Section 5.3) to the union of the Bloom filters $bf_{comb} := \bigcup_{j \in S} bf_j$, where S is the set of peers already selected for being queried, as the representation for the document space that has already been covered before. We define the novelty of Peer P_i (summarized in its Bloom filter bf_i) as

$$|\{k | bf_{comb}[k] = 0 \wedge bf_i[k] = 1\}|$$

i.e., as the number bits that are set in P_i's Bloom filter that are not yet set in bf_{comb}.

Analogously, we define the overlap between bf_{comb} and the Bloom filter of a peer P_i as

$$|\{k | bf_{comb}[k] = 1 \wedge bf_i[k] = 1\}|$$

i.e., the number of bits that are set in both Bloom filters. Note that this does not really yield the exact number of overlapping documents, but only a relative approximation that allows us to differentiate between several peers.

5.5 Discussion

The accuracy and, thus, the potential of our approach is directly related to the false positive probability introduced before, which in turn highly depends on the length of the Bloom filters. We currently assume that all Bloom filters across the system are of equal length, simplifying the bit operations necessary in our approach. The approach based on Bloom filters for estimating novelty was selected primarily due to its simplicity for implementation. We are currently in parallel developing alternative techniques based on min-wise independent permutations [4] for this orthogonal but important issue. With this length being a trade-off between accuracy on the one hand and bandwidth and storage consumption on the other hand, we have chosen the size of the Bloom filters so that they are adequately able to represent even the largest index lists in our experimental environment and leave determining the optimal filter sizes for future work, as it is outside the scope of this paper.

This issue becomes even more important bearing in mind that Bloom filters are sent along with each Post, i.e., one Bloom filter per $(Peer \times Term)$-pair.

6. PUTTING IT ALL TOGETHER

Similar to [28] we model relevance and overlap separately. To do so, we use a two-step approach that first ranks the collections according to an arbitrary quality estimator that yields numerical collection scores (like CORI) and subsequently re-ranks the collections incorporating their pair-wise overlap.

Algorithm 1 Algorithm to combine quality and overlap

```
1: input: s[i] and bf[i] for each peer i ∈ P = {1...n}
2: output: ranking rank[r] with all peers
3: S := P
4: j := argmax_{i∈S}{s[i]}
5: bf_comb := bf[j]
6: rank[1] := j
7: S := S\{j}
8: for r = 2 to n do
9:    quality[i] := computeQuality()
10:   novelty[i] := computeNovelty()
11:   j := argmax_{i∈S}{α · quality[i] + (1 − α) · novelty[i]}
12:   rank[r] := j
13:   S := S\{j}
14:   bf_comb := bf_comb ∪ bf[j]
15: end for
```

Algorithm 1 describes a simple combination of these two measures for quality and overlap. The computation works as follows: the algorithm has as input a set P of peers with scores $s[i]$ and Bloom filters $bf[i]$ for each of these peers (based on the quality estimation for a query). In the first step the top-ranked peer (with the highest score) is considered as a starting point promising high quality results and gets the highest rank. Note that we do not yet consider overlap, as the first collection will by definition contribute only novel documents, since no document has been seen before[2]. The further steps include picking the best peer of the remaining set of peers S in a loop. The computation of the best peer comprises the combination of quality and overlap of a peer i using a combination factor α:

$$\alpha \cdot quality[i] + (1 - \alpha) \cdot novelty[i]$$

Algorithm 2 Algorithm to compute the quality measure

```
1: input: s[i] for each peer i ∈ S = {1...m}
2: output: quality[i] for all peers i
3: s_max := max_{i∈S}{s[i]}
4: for i = 1 to m do
5:    quality[i] := s[i]/s_max
6: end for
```

Algorithm 2 describes the computation of the quality measure for all peers i in a set of peers. Note that in our main algorithm 1 the set S of peers decreases. The division by

[2]Yet this technique is easily applicable if we for example query the local collection first. In that case, the local peer is considered the top-ranked peer and the algorithm proceeds as usual

the maximum score s_{max} is used to normalize the original scores $s[i]$ of all remaining peers in S. This way $quality[i]$ of all peers is returned.

Algorithm 3 Algorithm to compute the contribution of a peer

```
1: input: bf_comb, bf[i] for each peer i ∈ S = {1...m}
2: output: overlap[i] for all peers i
3: for i = 1 to m do
4:    newDocs[i] := newDocs(bf[i], bf_comb)
5:    oldDocs[i] := oldDocs(bf[i], bf_comb)
6:    o[i] := newDocs[i]/log(oldDocs[i] + 2)
7: end for
8: o_max := max_{i∈S}{o[i]}
9: for i = 1 to m do
10:   novelty[i] := o[i]/o_max
11: end for
```

The computation of the overlap measure for all peers i is described in Algorithm 3. First, the algorithm computes a not normalized measure $o[i]$ by using the two functions *newDocs*() and *oldDocs*(). Both of them combine Bloom filters to estimate the number of new or old documents in $bf[i]$ in comparison to bf_{comb}. These two values are used to calculate $o[i]$ which increases with a higher number of new documents and with a lower number of old documents. In the second step, the algorithm takes the maximum value of $o[i]$ for normalization and returns $overlap[i]$ of all remaining peers.

The two functions *newDocs*() and *oldDocs*() pick up the idea of novelty in Section 5. The *newDocs*() function returns the number of set bits in the first Bloom filter which are not set in the second one:

$$newDocs(bf_1, bf_2) := |\{k | bf_1[k] = 1 \wedge bf_2[k] = 0\}|$$

The function *oldDocs* estimates the number of already seen documents and returns the number of bits that are set in both Bloom filters:

$$oldDocs(bf_1, bf_2) := |\{k | bf_1[k] = 1 \wedge bf_2[k] = 1\}|$$

Using this algorithm we get a refined ranking of all peers by considering quality and overlap. The combination factor α is variable between 0 and 1 to emphasize one of the two measures. In the case $\alpha = 1$, the algorithm just takes the original quality scores and delivers the same ranking of peers we would get considering only the quality measure.

7. EXPERIMENTS

7.1 Experimental Setup

One pivotal issue when designing our experiments was the absence of a standard benchmark. While there exist a number of benchmark collections for (centralized) Web search, it is not clear how to apply these collections to our scenario. While other studies partition these collections into smaller, disjoint pieces, we do not think this is an adequate approach. In contrast, we expect a certain degree of overlap among the collections, with popular documents being indexed by a substantial fraction of all peers, but, at the same time, with a large number of documents only indexed by a tiny fraction of all peers.

# of documents	62,962
# of features	9,254,996
max. length of index list	35,196

Table 1: Reference Collection Statistics

For this work, we have created a random sample of the GOV document collection [25] and partitioned it into disjoint fragments. Statistical details about this sample (subsequently also referred to as *reference collection*) are shown in Table 1. All recall measurements are relative to this reference collection. Collections were created by using various strategies to combine these fragments. For a crude estimate of the potential of our proposals, we have split our sample into six fragments and created collections by choosing all subsets with three fragments, thus, ending up with $\binom{6}{3}$=20 collections. For a more sophisticated analysis, we have split the sample into 100 fragments and used a variety of strategies to assign those to the collections:

- We randomly assigned/replicated each fragment to r peers.
- We select a fraction *alpha* of all peers as *hot*. Each fragment is assigned (replicated) r times, where each replica goes to a random hot peer with probability *beta* and to random cold peer with probability $1 - beta$.
- We use a *sliding window* over the fragments: the first peer receives r (consecutive) fragments f_1 to f_r, while the next peer receives the fragments from $f_{1+offset}$ to $f_{r+offset}$ and so on. By doing so, we can systematically control the overlap.

For the query workload we took all 50 queries from the topic-distillation track of the TREC 2003 Web Track benchmark [25].

All experiments were conducted on a prototype implementation of the MINERVA peer-to-peer web search engine described in Section 3. We order all collections according to their expected benefit to the query using CORI as a baseline and our proposed strategy. Preliminary experiments have shown that our algorithm produces the most robust results for values of *alpha* around 0.8; the exact choice of *alpha* is not critical, as the results for values of *alpha* between 0.6 and 0.9 showed only small differences. We therefore limit our experiments to this specific value.

In the experiments we compare the improvements in recall as we increase the number of peers that execute the query. Please note that, in all experiments, the recall when querying only one remote peer is equal for all strategies, as we currently don't take overlap into account when we have an empty result set representation, i.e. the first peer is chosen using the standard CORI method. The performance metric (consistent with our long-term goal to make distributed web search feasible) is the number of peers needed to be contacted in order to achieve a combined result of acceptable recall.

7.2 Experimental Results

7.2.1 $\binom{6}{3}$ Approach

We first conducted experiments on the 20 collections derived from subsets of three fragments each as described above.

As shown in Figure 2, our approach substantially outperforms CORI, that tends to select similar and, thus, highly overlapping peers. In order to reach a recall of 80%, for example, CORI needs to contact six peers, whereas our strategy can do the same with two peers. We would like to point out that considering overlap *only* in this environment would allow us to reach a recall of 100% with only 2 peers (because there are pairs of 2 peers each that cover the whole reference collection).

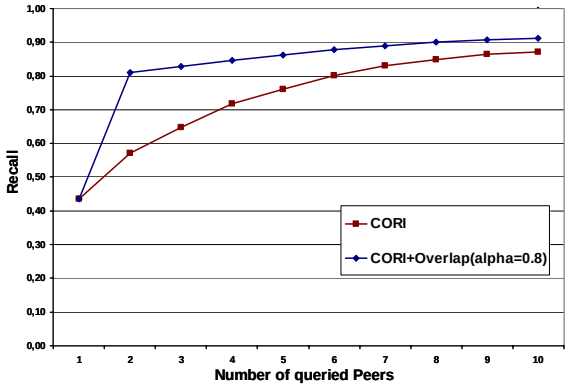

Figure 2: Recall on 20 Peers

Next, we additionally duplicated each peer, i.e., ending up with 40 peers. Theses results are presented in Figure 3. As expected, CORI now shows improvement at only every second peer, as each collection exists twice and is assigned adjacent ranks when estimating their expected quality. Our approach, however, shows continuous recall improvements, as the mirror collections are pushed down in the peer ranking. Even though this setting might seem fitted to our strategy, we argue that mirrored collections appear frequently in real-life, e.g., collections are duplicated in order to achieve fault-tolerance. This is particularly true in a P2P setting were replication is a mean to cope with the high dynamics of the system, as peers unpredictably enter and leave the system. In this setting, our approach again outperforms CORI in terms of both recall and number of peers required to achieve recall thresholds.(cf. Figure 3).

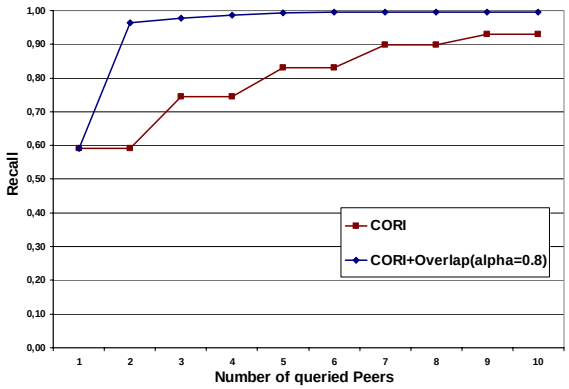

Figure 3: Recall on 2x20 Peers

7.2.2 Random Peers

On 20 peers created by randomly assigning each fragment to 4 peers, the improvements seen by our approach were smaller, because the pairwise overlap between the collections is almost negligible. Thus, this is a worst-case scenario for our algorithm. As can be seen in Figure 4, the improvement is typically less than 5%.

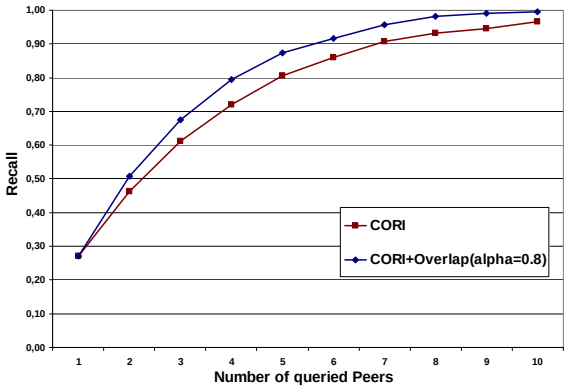

Figure 4: Recall on 20 Random Peers

7.2.3 Hot-Cold Approach

Next, we created peers using the *alpha*/*beta*-approach illustrated above. We chose $r = 4$, $alpha = 0.2$, and $beta = 0.8$ and created 40 distinct peers. The results closely resemble those obtained from the random collections, so we do not show a separate figure due to space constraints.

7.2.4 Sliding Window

Finally, we created peers using the notion of a sliding window as described above. We chose $r = 10$ and $offset$=2 to end up with 50 peers. Our approach again clearly outperforms CORI in this setting; in order to reach a recall of 80%, for example, CORI on average needs to contact 20 peers, whereas our strategy achieves this goal by contacting only 7 peers (Figure 5). It clearly shows substantial improvements already and especially for a relatively small number of peers. This underlines the importance of a powerful collection selection strategy in a distributed search environment, where efficiency in terms of bandwidth consumption and latency mainly depends on the number of contacted peers.

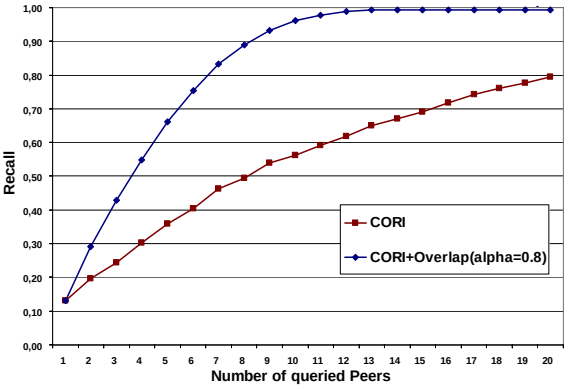

Figure 5: Recall on Sliding Window Peers

8. CONCLUSION

We have introduced a *query-specific* estimator of *mutual overlap* between collections and presented a novel way to

combine it with existing quality estimation metrics such as CORI. The experiments have proved the high potential of overlap-aware collection selection. It can drastically decrease the number of collections that have to be queried in order to achieve good recall. Depending on the actual degree of overlap between the collections, we have seen remarkable improvements especially at low numbers of queried peers. This fits exactly with our scenario of distributed web search where we want to put low limits in the number of peers involved in a query. We believe this is a large step toward making distributed, large-scale web search feasible.

We will extend our experiments to larger-scale data sets, not only in the field of text retrieval but also in the field of (semi-)structured data. We also plan to investigate the influence of different quality selection estimators other than CORI. Future work also includes further investigation on choosing the most efficient size of Bloom Filters, considering the trade-off between accuracy on the one hand and bandwidth and storage consumption on the other hand. Also, other estimators of mutual overlap will be studied for their performance and estimation characteristics. A more thorough analysis of the scalability and the actual resource consumption of our approach is also on our agenda.

9. 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] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[3] E. Buchmann and K. Böhm. 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*, Apr. 2004.

[4] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks. In Proceedings of ACM SIGCOMM, 2002.

[5] J. Callan. Distributed information retrieval. *Advances in information retrieval, Kluwer Academic Publishers.*, pages 127–150, 2000.

[6] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *SIGIR 1995*, pages 21–28. ACM Press, 1995.

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

[8] 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.

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

[10] D. Florescu, D. Koller, and A. Y. Levy. Using probabilistic information in data integration. In *The VLDB Journal*, pages 216–225, 1997.

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

[12] S. Ganguly, M. Garofalakis, and R. Rastogi. Processing set expressions over continuous update streams. In *SIGMOD 2003*.

[13] T. Grabs, K. Böhm, and H.-J. Schek. Powerdb-ir: information retrieval on top of a database cluster. In *CIKM 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] T. Hernandez and S. Kambhampati. Improving text collection selection with coverage and overlap statistics. pc-recommended poster. WWW 2005. Full version available at *http://rakaposhi.eas.asu.edu/thomas-www05-long.pdf*.

[16] 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.

[17] M. Mitzenmacher. Compressed bloom filters. *IEEE/ACM Trans. Netw.*, 10(5):604–612, 2002.

[18] Z. Nie, S. Kambhampati, and T. Hernandez. Bibfinder/statminer: Effectively mining and using coverage and overlap statistics in data integration. In *VLDB*, pages 1097–1100, 2003.

[19] H. Nottelmann and N. Fuhr. Evaluating different methods of estimating retrieval quality for resource selection. In *SIGIR, 2003*.

[20] 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.

[21] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Middleware 2001.

[22] L. Si, R. Jin, J. Callan, and P. Ogilvie. A language modeling framework for resource selection and results merging. In *CIKM 2002*.

[23] 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.

[24] 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.

[25] Text REtrieval Conference (TREC). http://trec.nist.gov/.

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

[27] J. Xu and W. B. Croft. Cluster-based language models for distributed retrieval. In *Research and Development in Information Retrieval*,1999.

[28] Y. Zhang, J. Callan, and T. Minka. Novelty and redundancy detection in adaptive filtering. In *SIGIR, 2002*.