# Query-Driven Indexing for Scalable Peer-to-Peer Text Retrieval[*]

Gleb Skobeltsyn[†], Toan Luu[†], Ivana Podnar Žarko[‡], Martin Rajman[†], Karl Aberer[†]

[†]Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
{gleb.skobeltsyn,vinhtoan.luu,martin.rajman,karl.aberer}@epfl.ch

[‡]University of Zagreb
Zagreb, Croatia
ivana.podnar@fer.hr

## ABSTRACT

We present a query-driven algorithm for the distributed indexing of large document collections within structured P2P networks. To cope with bandwidth consumption that has been identified as the major problem for the standard P2P approach with single term indexing, we leverage a distributed index that stores up to top-k document references only for carefully chosen indexing term combinations. In addition, since the number of possible term combinations extracted from a document collection can be very large, we propose to use query statistics to index only such combinations that are indeed frequently requested by the users. Thus, by avoiding the maintenance of superfluous indexing information, we achieve a substantial reduction in bandwidth and storage. A specific activation mechanism is applied to continuously update the indexing information according to changes in the query distribution, resulting in an efficient, constantly evolving query-driven indexing structure.

We show that the size of the index and the generated indexing/retrieval traffic remains manageable even for web-size document collections at the price of a marginal loss in precision for rare queries. Our theoretical analysis and experimental results provide convincing evidence about the feasibility of the query-driven indexing strategy for large scale P2P text retrieval. Moreover, our experiments confirm that the retrieval performance is only slightly lower than the one obtained with state-of-the-art centralized query engines.

**Categories and Subject Descriptors:** H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing – *Indexing Methods*; E.1 [Data Structures]: Distributed Data Structures

**General Terms:** Algorithms

**Keywords:** P2P, DHT, IR, Query-Driven index, Scalability

[*]The work presented in this paper was (partly) carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European projects BRICKS (507457) and ALVIS (002068).

## 1. INTRODUCTION

Extensive bandwidth consumption has been identified as one of the major obstacles for the adoption of peer-to-peer (P2P) technology in the field of information retrieval as recent studies [9, 23] have shown unscalable traffic requirements of web size document collections, even when sophisticated protocols are used to reduce retrieval costs. In our previous work, instead of indexing all single terms found in the document collection, which might lead to potentially very large posting lists and thus unscalable bandwidth consumption, we suggested an approach based on Highly Discriminative Keys (HDKs) [14]. This approach relies on indexing with terms and term combinations (hereafter called *keys*) that occur in at most $DF_{max}$ documents, where maximal document frequency $DF_{max}$ is a parameter of our model. Keys with a document frequency exceeding the predefined $DF_{max}$ threshold are associated with truncated posting lists, only storing top-$DF_{max}$ ranked documents and are expanded into possibly several larger keys (i.e., keys consisting of more indexing terms) with smaller document frequencies. Our scalability analysis [14] has shown that the number of generated keys grows linearly with the number of documents which is acceptable under a reasonable assumption that the ratio between the total number of documents and the total number of peers in the network remains bounded.

However, we have observed that the HDK approach generates a large number of keys that are never or rarely used in queries. Indeed, as the keys are generated only on the basis of their document frequencies, their popularity (and thus practical usefulness) is not taken into account. Obviously, the creation and maintenance of such superfluous keys causes substantial consumption of both bandwidth and storage, which represent valuable resources in large scale networks.

In this perspective, in parallel to our work on HDKs, we also designed the Distributed Cache Table (DCT) approach [17] that implements a purely query-driven indexing strategy. DCT's main assumption is that each peer is willing to provide only limited bandwidth and storage: it is therefore important to create an indexing structure that efficiently exploits the available resources. To do so, DCT generates an index on-the-fly by broadcasting queries that cannot be answered using the available indexing information and caches the obtained answers for future use. Only top-profitable query answers are cached, where profitability is defined to be proportional to the query popularity (or query frequency) and inversely proportional to its document frequency. To answer a query, a peer first looks up the distributed index for previously cached results that might answer the query. It is done by local filtering, as each cached result stores not only document references but also document digests containing the list of all distinct indexing terms per document. In particular, if the result for a query $q$ is cached, any query containing $q$ can be answered by local filtering of $q$'s result. As more and more queries get answered, the distributed cache repository progressively evolves, maintaining caches for popular and discriminative queries.

The main difference between the above mentioned indexing strategies (HDK and DCT) is in the process of handling rarely/never occurring queries. The HDK approach generates keys using solely the document collection, thus requiring no specific processing for queries containing rarely or never used keys, but at the price of substantial bandwidth and storage usage. Conversely, DCT uses both the document collection and the query history, and adapts the index to the current query popularity distribution thus yielding a more compact indexing structure. However, DCT requires expensive broadcasts to process rare queries and is rather suitable for middle size P2P text retrieval systems as it relies on local postprocessing of possibly large posting lists.

In this paper we describe a novel indexing strategy initially proposed in [18] that corresponds to a combination of the HDK and DCT indexing approaches and relies on popular highly-discriminative keys (pHDKs). The use of query statistics inherited from DCT leads to a substantial reduction of the generated key set. However, it does not require an immediate broadcast to answer a query, but relies only on the existing index to compute the query result. As a consequence, the quality of the result obtained for a given query depends on the popularity of the term combinations it contains. We have observed that this leads only to a marginal loss in retrieval quality, as the indexing structure constantly evolves by reacting to changes in the query distribution.

The paper is organized as follows: We describe the indexing/retrieval model in Section 2 followed by the algorithm description in Section 3. We then present the scalability analysis in Section 4 and the experiments in Section 5. We position our approach with respect to related work in Section 6 and provide a conclusion in Section 7.

## 2. DISTRIBUTED INDEXING/RETRIEVAL

Let us consider a structured P2P network with $N$ peers $P_i$, $1 \leq i \leq N$, and a possibly very large document collection $\mathcal{D}$, consisting of $M$ documents $d_j$, $1 \leq j \leq M$. $M$ is referred to as the size of $\mathcal{D}$, while $T_{\mathcal{D}}$ is the term vocabulary in $\mathcal{D}$ and $|T_{\mathcal{D}}|$ is used to denote the number of terms in $T_{\mathcal{D}}$.

In addition, we assume that a large query log $L$ is available, where each query $q \in L$ is a set of terms. The set of all terms present in the query log is denoted by $T_L$, and the result of the intersection $T_L \cap T_{\mathcal{D}}$ corresponds to the terms used in meaningful queries (producing non-empty results).

In the P2P network, each peer $P_i$ plays two complementary roles. First, $P_i$ stores a fraction of the global document collection $\mathcal{D}$, denoted by $\mathcal{D}_i$, and builds a local index for $\mathcal{D}_i$. Second, $P_i$ contributes to store and maintain the global inverted index that associates indexing keys (i.e., indexing term sets) to documents in $\mathcal{D}$. The fraction of the global index under the responsibility of $P_i$ consists of all the keys and associated posting lists (i.e., document references) that are allocated to $P_i$ by the Distributed Hash Table (DHT) built on top of the P2P network.

As far as indexing is concerned, each peer $P_i$ is responsible for the following complementary tasks:

- First, $P_i$ is responsible for indexing $\mathcal{D}_i$, i.e., for computing the indexing keys and associated posting lists that can be locally derived from $\mathcal{D}_i$, and for inserting them into the global P2P index.
- Second, $P_i$ is responsible for maintaining its fraction of the global index. More precisely, $P_i$ maintains pairs of the form $(k,\ PL(k))$, where $k$ is an indexing key that $P_i$ is responsible for and $PL(k) = \{d_j \in \mathcal{D} \mid k \in d_j\}$ is the posting list associated with $k$. Notice that a $(k,\ PL(k))$ pair stored in the fraction of the global index under the responsibility of $P_i$ has no a priori reason to be the one that $P_i$ extracts from $\mathcal{D}_i$.
- And third, $P_i$ maintains a fraction of the global query statistics for all terms and term combinations it is responsible for.

As far as retrieval is concerned, each peer $P_i$ is responsible for the following complementary tasks:

- First, while processing a query $q$, $P_i$ interacts with the P2P network in order to retrieve and rank the list of documents in $\mathcal{D}$ that contain indexing keys that maximally overlap with the set of terms $q$ consists of.
- Second, $P_i$ triggers the update of the query statistics for the queries it is processing.
- Third, based on the updated query statistics, $P_i$ might initiate the key activation mechanism described in Section 2.2.

A detailed description of the above-mentioned indexing and retrieval models is given below.

### 2.1 Indexing/Retrieval Model

The goal of distributed indexing is to generate and maintain a suitable set of indexing keys associated with the corresponding global posting lists. Since the indexing process is computationally intensive, peers share the indexing load to collaboratively build the required distributed index.

First, the peers build the global single-term index for $\mathcal{D}$. To do so, each peer $P_i$ performs local indexing, i.e., it extracts all single-term keys that can be found in its local collection and inserts them into the P2P network, along with the corresponding posting lists, possibly truncated to their top-$DF_{max}$ ranked elements if their size exceeds the predefined $DF_{max}$ threshold. We call this index the *basic single-term index* as it enables the basic query processing functionality.

The following document indexing process is fully driven by the query processing, and is performed in parallel with retrieval. More precisely, as soon as a peer receives a new query $q$, it starts to explore the lattice of the query term subsets in a top-down fashion (hereafter simply called the *query lattice*), in decreasing subset size order, starting with the query itself. An example query lattice is given in Figure 1. For each explored query term subset $q' \subseteq q$ (hereafter called a *query key*), the peer interacts with the P2P network to check whether the query key $q'$ is already associated with an existing posting list, and to update (and retrieve) its estimated probability of use $EPU(q')$. $EPU(q')$ is computed

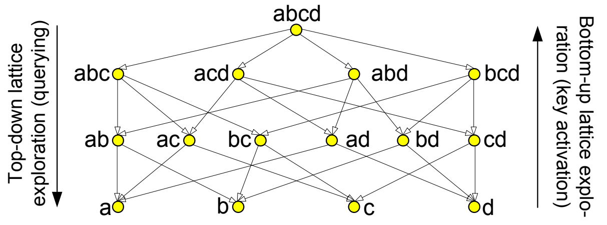
Top-down lattice exploration (querying)
abcd
abc
acd
abd
bcd
ab
ac
bc
ad
bd
cd
a
b
c
d
Bottom-up lattice exploration (key activation)

**Figure 1: Query lattice exploration (an example)**

for the term combination $q'$ based on its usage frequency statistics.

If a posting list is retrieved from the P2P network, it is locally stored, and the part of the query term lattice dominated by the query key $q'$ (i.e., all term subsets contained in $q'$) is excluded from the lattice exploration process (see Section 2.4).

Once the top-down query lattice exploration process is completed, all the retrieved posting lists are cumulated, reranked, and the top-ranking document references are presented to the user as a result to the submitted query.

Subsequently, the bottom-up query lattice exploration is initiated starting with the query keys of size 2. For each explored query key $q'$ that is not yet associated with a posting list[1], if the estimated probability of use $EPU(q')$ exceeds the predefined threshold $EPU_{min}$, the peer triggers the production of the posting list to be associated with $q'$ (we say that the query key $q'$ is *activated*) by starting the *opportunistic notification mechanism* described in Section 2.3. A query term subset $q'$ is activated only if all direct descendants of $q'$ (all immediate subsets of $q'$) are already associated with *truncated* posting lists due to redundancy filtering as explained in Section 2.4.

To summarize, there are two independent processes constantly running at each peer in the network:

- *Document indexing*: Each peer takes care that its local document collection is properly represented in the global distributed index. As the content of the global index is driven by the query-load, it is constantly evolving. Hence, when the document indexing module is notified that a new term combination should be indexed (a new key was activated), it queries its local document collection for the combination and reports the result (if any) to the peer responsible for the key.

- *Query processing*: Since every peer maintains a portion of the global index, it might be requested to provide a part of it or/and update the popularity statistics. The query initiating peer, however, coordinates the query processing as it is interested in obtaining the final result.

## 2.2 Query-driven key activation

Since querying and indexing are in general two independent processes, the implementation of the query-driven key generation is complex as it requires the knowledge about query statistics that might not be available for a given key at the moment when the document matching the key is being indexed. To cope with this asynchronism, an activation mechanism is employed that triggers the generation of a new key only when its estimated probability of use reaches a certain threshold.

*Definition 1.* A key $k$ is *non-superfluous* iff 1) its size $\geq 2$ and 2) $EPU(k) \geq EPU_{min}$, where $EPU(k)$ is the estimated probability of use computed for the key $k$.

Indeed, as the quality of query results provided by the basic single-term index is not sufficient, we are interested in populating the rest of the index with non-superfluous keys, ensuring that all such keys are actually used during query processing. We also apply the size and redundancy filters as described in Section 2.4 to further optimize the index.

During the retrieval process, a key of size $\geq 2$ can be identified as popular based on its usage statistics and has to be "activated". It can be used for query answering when the associated global posting list is generated and stored in the network. To do so, an "indexing request" is sent to all the peers that hold documents containing the activated key. However, as the exact list of such peers is unknown at activation time, to efficiently propagate the indexing request we use the *opportunistic notification mechanism*, described in Section 2.3. Upon receiving the indexing request for an activated key $k$, each peer queries its local index for $k$ and, provided the result is non-empty, sends it to the peer responsible for $k$.

Thus, the processing of new queries leads to the activation of new keys, which, in turn, results in better result quality for subsequent queries.

## 2.3 Opportunistic Notification Mechanism

The opportunistic notification mechanism (ONM) is used to inform all peers in the network that a certain key has been activated. Essentially, ONM is a bandwidth-friendly version of broadcast. It has several distinguishing features described below that considerably reduce the bandwidth consumed by the key activation/indexing process.

We assume the underlying P2P network supports a shower broadcast algorithm as described in [5] that guarantees that each peer receives at most one and sends at most $\log N$ messages, where $N$ is the number of peers in the network. Moreover, the overall latency of such a broadcasting mechanism is $O(\log N)$.

ONM messages have low priority and small size, hence, we use the so-called "hitchhiking" (or piggybacking) mechanism to propagate them. As we can tolerate a certain delay to accomplish the indexing request dissemination, every ONM message is not sent immediately to the next-hop peer, but waits for another higher priority message to reach the same destination. Such an event is likely to happen, as routing or maintenance messages are being frequently sent between neighboring peers. If such a "carrier" message is detected within a $TTL_{ONM}$ period, the notification information is appended to the high-priority message. We call it hitchhiking because the ONM message uses the available space of the carrier's TCP/IP packet. After $TTL_{ONM}$ expires, the message is sent directly.

Instead of broadcasting the request to all peers at once, the ONM is split in several multicast sessions. When a key is activated, the request is propagated only to the first random fraction of peers. After a random timeout, or when the popularity of the key further increases, the next fraction of peers is contacted and so on until all the peers are notified.

[1]Recall that all single terms are initially associated with (possibly truncated) posting lists.

The shower mechanism (shower multicast) can still be applied for each session separately if the fraction of peers is specified by a continuous key range [5]. Notice that even partial posting lists obtained from the first session have the potential to improve precision. As a result of this notification process performed in iterations, the bandwidth load is distributed over time.

The indexing traffic generated by peers responding with their local indexing information for activated keys can be reduced using the following principle. If the relevance of a document to the activated key is low, it might not appear among the top-$DF_{max}$ records and the indexing peer should avoid the transmission of documents with relatively low scores to the peer responsible for the key. Thus, in the subsequent multicast sessions, a current rank threshold[2], denoted as *minRank*, is also propagated, causing pruning of documents ranked lower than the announced threshold. Thus, we reduce the bandwidth consumption and the load on the peer responsible for the activated key. The reduction is substantial for keys that occur in a large number of documents.

## 2.4 Document-driven key filtering

Apart from determining the potential of indexing keys from the query distribution, the indexing procedure is based on the principle of choosing the adequate keys from documents based on the HDK approach originally introduced in [14]. The HDK approach is used to generate a key set of a scalable size from the document collection, based only on document frequencies. In this work, we are rather interested in the set of keys that appear in the intersection between the exhaustive set of popular keys obtained from the query-log and the set of keys generated using the HDK indexing principle. The idea behind the HDK approach is quite intuitive and is depicted in Figure 2. The generated set of keys is based on selected terms and term sets that occur in at most $DF_{max}$ documents, where $DF_{max}$ is a parameter of our model. The crucial characteristic of this indexing method is that it leads to an increase in the total number of index entries, but, at the same time, limits the size of the associated posting lists to $DF_{max}$, which strictly bounds the traffic generated during retrieval. This approach is fully in line with the general properties of P2P networks that can easily store large amounts of data (provided that enough peers are available), but must be carefully controlled with respect to the volume of information transmitted between the peers.

In this section we outline several filtering techniques we apply to reduce the number of keys based on their distribution in the document collection. For a more detailed description of the filtering techniques please refer to [14]. Notice that a peer applies these filters to prune the activation of redundant keys and requires no knowledge about the query statistics.

**Size filtering** limits the size of an indexing key to a maximal size $s_{max}$. For example, in the case of web retrieval, the average query size is currently estimated to be between 2 and 3 terms. Notice that limiting the size of the candidate keys does not have any substantial impact on the global indexing quality because, for a well chosen value of $s_{max}$, most of the user queries would have a size smaller than or equal to $s_{max}$. For a few queries of size bigger than $s_{max}$, the retrieval mechanism described in Section 3.2 is applied. Obviously, setting $s_{max} = \infty$ disables the size filtering and can also be handled by our approach.

[2]Rank threshold is the score of the $DF_{max}$-st document from the already obtained partial result.

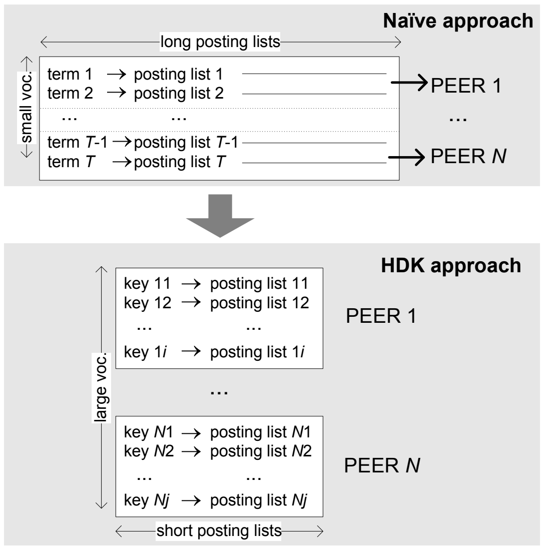

**Figure 2: The basic idea of HDK indexing**

**Discriminative and non-discriminative keys.** Each key $k \in K$ is associated with its document frequency $df(k)$ corresponding to the number of documents in the collection $\mathcal{D}$ that contain $k$.

Given a document frequency threshold $DF_{max}$ such that $1 \leq DF_{max} \leq M$, we use the key document frequencies to classify the keys into two distinct categories: *discriminative* keys and *non-discriminative* keys.

*Definition 2.* $K_d = \{k \in K \mid df(k) \leq DF_{max}\}$ is the set of *discriminative keys* (DKs), i.e., the keys that appear in at most $DF_{max}$ documents and therefore have a high discriminative power w.r.t $\mathcal{D}$.

*Definition 3.* $K_{nd} = \{k \in K \mid df(k) > DF_{max}\}$ is the set of *non-discriminative keys* (NDKs), i.e., the keys with low discriminative power w.r.t $\mathcal{D}$.

Notice that the DKs (resp. NDKs) verify the following *subsumption property*: Any key containing a DK of smaller size is also a DK. Respectively, any key contained in an NDK of bigger size is also an NDK.

**Redundancy filtering.** This filtering method relies on the subsumption property of the DKs to further reduce the number of candidate keys. If a key $k_1$ contains a discriminative key $k_2$ of a smaller size, then $k_1$ is also discriminative and the answer set $PL(k_1)$, which is contained in $PL(k_2)$, can be produced by local postprocessing of $PL(k_2)$. In other words, $k_1$ is practically redundant with respect to $k_2$ and therefore does not need to be stored in the global index.

*Definition 4.* A key $k$ is *highly discriminative* iff 1) $|k| \leq s_{max}$ (size filtering); 2) $k$ is discriminative; and 3) all its subkeys of strictly smaller size are non-discriminative. In the rest of this paper, highly discriminative keys will be referred to as HDKs.

In other words, redundancy-based filtering implies considering only highly-discriminative keys and all their (non-discriminative) sub-keys for indexing. It greatly reduces the number of candidate keys, but, due to the subsumption property, fully preserves the indexing exhaustiveness.

## 3. INDEXING/RETRIEVAL ALGORITHMS

### 3.1 Indexing

The goal of the query-driven indexing (QDI) algorithm is to maintain, for the global document collection $\mathcal{D}$ distributed over $N$ peers, Single-Term Keys (STKs), popular NDKs and HDKs (pNDKs and pHDKs), and to associate them with the corresponding global posting lists. Exhaustive posting lists are stored for pHDKs, while for pNDKs and for non-discriminative STKs posting lists are truncated to their top-$DF_{max}$ elements.

Algorithm 1 describes the *indexDocuments* routine that is executed by every peer when it is notified that a new key $k$ has been activated. The peer executes a lookup over its local document collection (line 1) and, if the result is not empty, sends the list of documents with scores above $minRank$ to the peer responsible for the key (lines 3–4).

**Algorithm 1** $indexDocuments(k,\ minRank)$

```
1: $result \leftarrow LocalQueryLookup(k,\ minRank)$;
2: **if** $result \neq null$ **then**
3:    $peer = DHT.route(generateKey(k))$;
4:    $peer.updatePostingList(k,\ result)$;
5: **end if**
```

However, a peer responsible for a frequent key can be overloaded with the number of incoming messages. This situation is efficiently resolved by the DHT-level congestion control module [8] developed in our group. Moreover, this module takes care of the on-the-fly aggregation of the messages with the same next-hop destination. Also, note that the ONM mechanism would further reduce the indexing traffic by propagating the notification in several rounds and by communicating the rank value threshold ($minRank$) in each subsequent round as described in Section 2.3.

### 3.2 Retrieval

The goals of the retrieval algorithm are: 1) to locate, for a given query $q = \{t_1, t_2, \ldots, t_{|q|}\}$, the corresponding keys in the global P2P index, 2) to retrieve the posting lists associated with the keys, 3) to rank the union of obtained posting lists, and 4) to output the computed top-k result to the user.

To process a query, the query originator first probes all peers responsible for all keys of size $min(s_{max}, |q|)$ extracted from the query (see the loop in lines 4–23 in Algorithm 2). Recall that it is done by hashing a string representation of each term combination extracted from the query. The popularity statistics for each key are updated at these peers (line 8) and for already activated (indexed) keys their posting lists are located (line 10). Then, term combinations of smaller sizes are processed in the same way with one exception: no sub-combinations of a combination that was discovered before as indexed have to be considered (line 6). E.g., if we are processing a query $\{a, b, c, d\}$ and the index item for $\{a, b, c\}$ is found, there is no need to process any sub-combination of $\{a, b, c\}$ such as $\{a, b\}$, $\{b, c\}$, $\{a, c\}$, $\{a\}$, $\{b\}$ and $\{c\}$, since they would not provide any new relevant results. Generally, the union of obtained results provides the list of peers that have to be contacted to obtain the final, properly ranked result. Notice that due to the set containment property, whenever a posting list of a size below $DF_{max}$ for a key $k \subseteq q$ is found, the final result for the query $q$ is contained in the exhaustive posting list of $k$.

While processing a query $q$, the estimated probability of use for some key $k \subseteq q$ might exceed the $EPU_{min}$ threshold. In case all sub-combinations of size $|k| - 1$ are already indexed and their posting lists are truncated to $DF_{max}$ postings, $k$ is activated and the ONM is executed to announce the new key (line 24). If at least one sub-key $k' \subset k$ of size $|k| - 1$ does not exist, it has to be created first, as its $EPU(k')$ is guaranteed not to be smaller than $EPU(k)$. Alternatively, if there exists any indexed key of size $|k| - 1$ that is associated with a posting list smaller than $DF_{max}$, no extra key has to be activated.

**Algorithm 2** $processQuery(q)$

```
1: $result \leftarrow \emptyset$;              /*stores obtained document identifiers*/
2: $found \leftarrow \emptyset$;       /*stores keys that were identified as indexed*/
3: $candidates \leftarrow \emptyset$;     /*stores keys that will be activated after*/

4: **for** $i = min(s_{max}, |q|)$ downto 1 **do**
5:    **for** $k \leftarrow generateNextTermCombination(q,\ i)$ **do**
6:       **if** $\forall s \in 1..|found|,\ k \nsubseteq found[s]$ **then**

            /*route to the peer and increase the popularity of $k$:   */
7:          $peer = DHT.route(generateKey(k))$;
8:          $peer.incPopularity(k)$;

            /*$k$ is indexed ⇒ request its posting list, add to $result$:*/
9:          **if** $peer.getIndexState(k)$ = INDEXED **then**
10:            $result \leftarrow result \cup peer.getResult(k)$;
11:            $found \leftarrow found \cup k$;
12:         **end if**

            /*$k$ is not indexed and popular ⇒ add to $candidates$: */
13:         **if** ($peer.getIndexState(k)$ = NOT-INDEXED)
            **and** ($peer.getPopularity(k) > EPU_{min}$) **then**
14:            $candidates \leftarrow candidates \cup k$;
15:         **end if**

            /*$k$ is not indexed OR $k$ is a discriminative key ⇒     */
            /*delete all already found candidates that contain $k$:  */
16:         **if** ($peer.getIndexState(k)$ = NOT-INDEXED)
            **or** ($peer.getFrequency(k) \leq DF_{max}$) **then**
17:            **while** ($\exists k' \in candidates,\ s.t.\ k \subset k'$) **do**
18:               $candidates \leftarrow candidates \ \setminus\ k'$;
19:            **end while**
20:         **end if**

21:      **end if**
22:   **end for**
23: **end for**

24: $DHT.executeONM(candidates)$; /*index new keys if any*/
25: **return** $result$;
```

In order to estimate $EPU(q) = qf(q)/|L|$, where $qf(q)$ denotes the query frequency for $q$, each peer should be able to estimate the global number of queries $|L|$, which can be propagated among the peers, e.g., using [1]. Alternatively $EPU(q)$ can be estimated by maintaining a number of recent "hits" during a globally known period of time, though it would require a slightly more complicated counting facility. Note that considering recent statistics facilitates timely reaction to changes in the query popularity distribution.

## 4. SCALABILITY

The QDI approach presented in this paper is scalable in terms of bandwidth consumption because: 1) the retrieval traffic generated while processing a query is low, since all transmitted posting lists are of size limited to $DF_{max}$, 2) the indexing traffic generated to activate a key (the ONM traffic) and populate the global posting list is scalable.

The amount of *retrieval traffic* generated while answering a query $q$ is bounded by

$$RT(q) = DF_{max} \sum_{i=1}^{min(s_{max},|q|)} \binom{|q|}{i}$$

that depends only on the size of the query and the $s_{max}$ parameter.

The number of indexed keys determines the amount of *indexing traffic* sent through the network while populating the index. Indeed, activation of a key $k$ requires execution of the ONM that transmits at most $b_a N$ bytes, where $b_a$ is the size of the transmitted notification, which contains the key itself and the *minRank* value. Once, the ONM is executed, indexing a key $k$ consumes at most $b_i df(k)$ bytes, where $df(k)$ is the document frequency of $k$ and $b_i$ is the size of transmitted indexing information including an IP-address, a port number, and a document reference. Therefore, the upper bound for the indexing traffic generated for a key $k$ can be estimated as:

$$IT(k) = b_a N + b_i df(k),$$

where the first part grows linearly with the number of peers and the second part grows linearly with the size of the document collection.

Hence, the number of indexed keys is crucial to ensure the indexing traffic scalability.

In [14] we showed that the number of keys generated for a given document collection by applying the HDK indexing described in Section 2.4 grows linearly with the collection size. The query-driven key activation mechanism results in a substantial decrease of the number of keys as it can be viewed as an additional filter based on the query distribution properties that eliminates superfluous keys. Here, we derive an upper bound on the number of keys that can be generated from the queries contained in the query log and show that it scales linearly with the query log size. Since we cannot capture analytically the correlation between the term combination popularity and its document frequency, the upper bounds are computed based on the query distribution properties only. Moreover, in Section 5.1 we will experimentally show that despite of significant reduction of the number of indexed keys, the QDI approach causes only a marginal loss of the answer quality even for web-size document collections and real query distributions.

If we abandon the correlation between the term combination popularity and its posting list size, the number of possible combinations selected for indexing depends only on the query log properties. Our analysis of real query distributions shows that the number of term combinations found in the query log follows the Pareto [15] distribution. We use this assumption to analyze the number of keys that can be potentially generated from the queries appearing in the recent query log. This number, however, is an upper bound and is significantly reduced in practice by applying the data-driven key filtering.

We are interested in deriving the number of combinations with minimum query frequency (popularity) $QF_{min}$ after $|L|$ queries were observed in the query log. Recall, that the number of single term keys does not depend on the query distribution as all terms found in the document collection are indexed. According to the Heaps' law [7], the number of distinct terms grows as $O(\sqrt{n})$ with the size of the document collection $n$, and, therefore can be practically managed in a P2P network. For our future analysis, we ignore single term keys and concentrate on combinations of size 2 and larger.

We assume that frequencies of keys of sizes $2, ..., s_{max}$ are Pareto distributed, i.e., the probability that the query frequency $qf_k$ of a key $k$ is higher or equal to a predefined $QF_{min}$ is given by

$$P(qf_k \geq QF_{min}) = QF_{min}^{-\alpha}, \ |k| \geq 2,$$

where $\alpha > 0$ denotes the Pareto index.

Note that since $qf_k = |L| \ EPU_k$ and $QF_{min} = |L| \ EPU_{min}$, the expression above can be rewritten as:

$$P(EPU_k \geq EPU_{min}) = (|L| \ EPU_{min})^{-\alpha},$$

where $EPU_k$ denotes the estimated probability of use for $k$.

We define $|\mathcal{K}_1|$ as the number of keys that have appeared only once in the query log. The total number of combinations $\gamma$ found in a query log $L$ is thus

$$\gamma = |\mathcal{K}_1| \int_1^{+\infty} (QF_{min})^{-\alpha} \ d(QF_{min}).$$

Assuming[3] $\alpha > 1$:

$$\gamma = \frac{|\mathcal{K}_1|}{\alpha - 1}$$

or

$$|\mathcal{K}_1| = \gamma \ (\alpha - 1).$$

The number of keys to be indexed for the chosen $QF_{min}$ is the following:

$$|\mathcal{K}_{QF_{min}}| = |\mathcal{K}_1| \ P(qf \geq QF_{min}) =$$

$$= (\alpha - 1) \ (QF_{min})^{-\alpha} \ \gamma = O(\gamma).$$

Assuming the number of term combinations generated for a query is bounded by a constant calculated from the maximum query size, we can write

$$|\mathcal{K}_{QF_{min}}| = O(|L|).$$

Therefore, the number of indexed keys grows linearly with the number of queries submitted to the system. Moreover, if we increase the $QF_{min}$ parameter, the proportionality factor $(\alpha - 1) \ (QF_{min})^{-\alpha}$ falls exponentially, and the constant is rather small for reasonably chosen $QF_{min}$.

In [14] we have shown that key generation based on the HDK approach produces the key set $K_{\mathcal{D}}$ of size $|K_{\mathcal{D}}|$, which grows linearly with the document collection size. On the other hand, the upper bound for the number of keys extracted from the queries $|\mathcal{K}_{QF_{min}}|$ grows linearly with the query log size. In practice, as we combine the query-driven key generation with the document-driven filtering the total number of generated keys is the size of intersection of the two sets: $|K_{\mathcal{D}} \cap \mathcal{K}_{QF_{min}}|$. Indeed, by removing all superfluous keys from $K_{\mathcal{D}}$, we obtain $\mathcal{K}_{QF_{min}}$.

Thus, we have shown that the number of generated keys and, hence, the total generated traffic is indeed scalable.

[3]Our experiments suggest values of $\alpha$ being close to 2.

## 5. EXPERIMENTS

We showed that the QDI approach scales to web sizes with respect to the size of the index. However, by pruning superfluous keys we have to tolerate a certain degradation of the retrieval quality. Since we cannot capture analytically the correlation between the popularity of a certain term combination and its document frequency, we conducted a large-scale experiment described in Section 5.1 that shows that the retrieval quality remains acceptable for reasonably chosen $EPU_{min}$.

In Section 5.2 we also report obtained simulation results. In particular, we observed a significant bandwidth consumption decrease when compared with the HDK approach.

### 5.1 Overlap experiment

For our experiments we used a real query log from the Wikipedia online encyclopedia [22], which contains more than $9M$ queries logged during September and October 2004. We used this query log to implement a query generator that produces random query sets following the true observed query popularity distribution. We then generated a test set of 3000 queries and, for each of these queries, built a reference answer set by retrieving the top-20 results produced by the Google search engine.

We also extracted all the term combinations present in the query log and evaluated their estimated probability of use. The query log contained more than $9M$ queries and we observed around $10M$ unique term combinations. Notice that these combinations form the set of all non-superfluous keys $\mathcal{K}_1$ corresponding to $QF_{min} = 1$.[4] For values of $QF_{min} \geq 1$, we can obtain $\mathcal{K}_{QF_{min}}$ by filtering out keys with query frequency below $QF_{min}$.

Before running our experiments, the (distributed) single term index was generated by retrieving top-$DF_{max}$ Google answers for each single term. Afterwards, all $9M$ queries were processed by our distributed retrieval engine to activate popular keys. The final content of the distributed index therefore consisted of all the keys in $\mathcal{K}_{QF_{min}}$, along with all the single terms in $T_{\mathcal{D}}$. Notice that the activation process is sensitive to the $QF_{min}$ parameter.

To evaluate the quality of the answer set produced by our query-driven retrieval engine for a query $q$, we compared the union of the posting lists associated with all the keys contained in $q$ and present in the distributed index with the reference answer set produced for $q$ by the Google search engine. More precisely, for each of the queries, we measured the overlap between the produced answer and the reference result set. E.g., an overlap of 100% means that the produced answer set contains all the documents present in the reference set. The $DF_{max}$ parameter was set to 100.

Figure 3-a shows the achieved overlap values for different $QF_{min}$ and $s_{max}$. Figure 3-b shows the fractions of the queries corresponding to full, partial or no overlap when $s_{max}$ is set to 3.

Some interesting observations can be made from these plots: 1) There is almost no benefit in considering keys of size $\geq 3$ (notice however that our approach can efficiently handle keys of any size); 2) Good overlap can be obtained for reasonable values of $QF_{min}$ (e.g. an overlap of 80% for $QF_{min}$ set to 8); 3) The 80% overlap corresponds to 6% of the queries with no intersection with the reference set.

[4]$QF_{min} = x$ for a 9M query log implies $EPU_{min} = x/9M$.

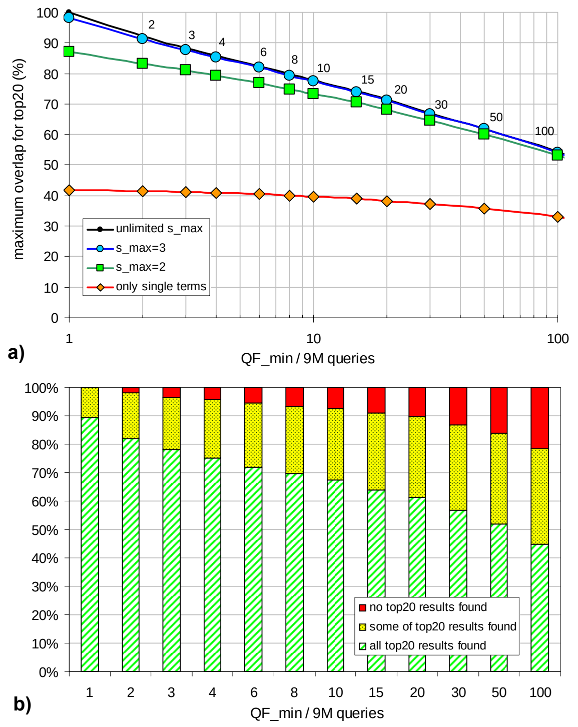

Figure 3: The overlap obtained with Google.

We performed the same experiment using the Yahoo search engine and obtained very similar results. This is an additional evidence that our query-driven approach delivers good quality retrieval for real web-size document collections. However, we admit that Wikipedia logs are quite specific as users usually search for concrete articles in the encyclopedia. Thus, our results could be slightly worse for real search engine logs. Investigation of this issue is an important part of our future work.

### 5.2 P2P index simulations

In this section we analyze the performance of our approach in a dynamic setting. Starting from the basic single-term index, we observe how processing of new queries triggers indexing of newly activated keys and improves the retrieval quality. We conducted a set of experiments with the Wikipedia document collection (containing 650K articles) and the query log mentioned in Section 5.1. The experiments were carried out with the following parameters: $DF_{max} = 100$, $EPU_{min} = \frac{4}{2M}$ (i.e., a key is considered non-superfluous if it occurs at least 4 times among the 2M recent queries) and $s_{max} = 3$.

Figure 4 shows the number of generated keys as queries are being processed. Notice that, out of $1.3M$ single-term keys, less then $200K$ were actually used in queries, and that the total number of generated keys was reduced by $1 - 2$ orders of magnitude when compared to the HDK approach without the query-driven key activation. We estimated that the HDK approach would produce around $65M$ keys in this scenario, whereas our approach requires only $0.5M$ keys to be activated with $EPU_{min} = 2/4M$, in addition to the $1.3M$ single-term keys, while, as shown in Figure 5, the retrieval quality remains reasonable.

Based on the experimental data we will show now that the QDI approach generates less traffic than the HDK ap-

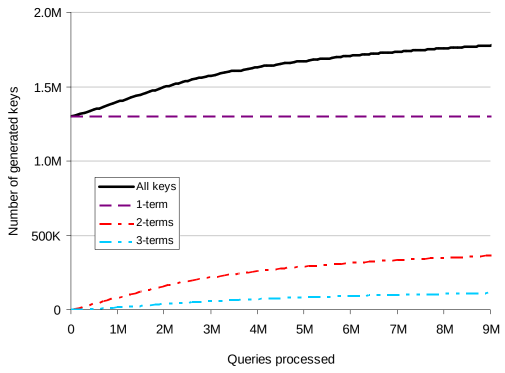

**Figure 4: Number of generated indexing keys as a function of the number of processed queries.**

proach, which was proven to be scalable with respect to bandwidth consumption during indexing. To do so, we estimate the indexing traffic of the HDK approach as the sum of document frequencies for all keys in the index $K_{\mathcal{D}}$: $IT_{HDK} = b_i \sum_{|K_{\mathcal{D}}|} df(k)$, where $b_i$ is the size of the transmitted indexing information. The QDI approach would consume $IT_{QDI+ONM} = b_i \sum_{|K_{QDI}|} df(k) + b_a N K_{QF_{min}}$, where the first part corresponds to the actual indexing traffic and the second one reflects the extra traffic consumed by the ONM executions. As the number of keys is significantly smaller compared to the HDK approach, QDI requires less indexing traffic. As the number of peers grows, the ONM mechanism becomes more and more expensive. However, as the number of peers is correlated to the size of the document collection, i.e., each new peer adds some constant number of new documents in the collection, the QDI approach would induce significantly lower traffic. Moreover, the bandwidth-saving features of the ONM algorithm would further reduce the bandwidth consumption.

Let us make a simple example with real numbers. Our test collection contains 650K documents. Each document contains 128 distinct terms on average. If we set up $b_i$ to 25 bytes, the generation of the single term index would require $IT_{ST} = 650K \cdot 128 \cdot 25 \approx 2$ Gb of indexing traffic. The HDK approach generates roughly 1600 keys per document, resulting in $IT_{HDK} = 650K \cdot 1600 \cdot 25 \approx 26$Gb. The QDI approach prunes all superfluous keys and indexes only $0.5M/65M \approx 0.8\%$ of the keys of size 2 and 3 when compared to the HDK approach. Thus, $IT_{QDI} = IT_{ST} + 0.008 \cdot 26$Gb $\approx 2.2$Gb. Therefore, $IT_{QDI}$ is an order of magnitude smaller than in the case of the HDK approach. The price we pay is the extra ONM traffic estimated as $IT_{ONM} = 10 \cdot 0.5M \cdot N = 5 \cdot N$ Mb, assuming we pack an ONM notification in 10 bytes. For instance, if each peer stores 65K documents, the whole collection of 650K documents will be distributed at around 100 peers. Thus, the total indexing traffic generated by the QDI approach would be $2.2GB + 5M \cdot 100 \approx 2.7$ Gb which is still an order of magnitude smaller then the traffic generated by the HDK approach.

We plan to provide more experimental evidence on the bandwidth scalability of the QDI approach with the prototype we are currently developing.

We also measured the average overlap for the query results obtained with our approach compared to the full single-term index based on the Terrier retrieval engine [21]. With the same evaluation methodology as for the overlap experiment described in section 5.1, the obtained overlap values are shown in Figure 5. These values show that the retrieval quality grows quite fast with the number of processed queries, starting from a relatively low value corresponding to the single term index. At each point the overlap value was obtained by processing of a test set of 50K queries with a given state of the index. Index updates were frozen during the overlap measurements. For comparison we show similar plot obtained for $EPU_{min} = \frac{3}{2M}$.

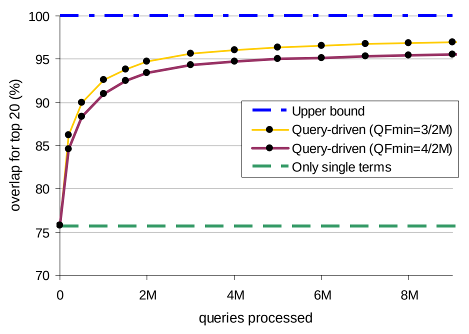

**Figure 5: Average overlap for the index formed after processing a number of queries.**

Figure 6 shows the upper bounds for the overlap that can be achieved with our indexing mechanism for different $EPU_{min}$ values. This figure is similar to Figure 3, but, as the document collection is much smaller shows higher overlaps.

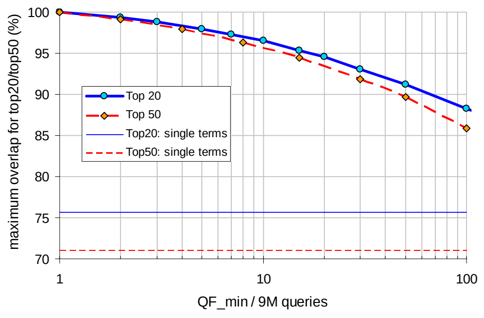

**Figure 6: Overlap upper bound for the Wikipedia document collection with different $EPU_{min}$.**

The plots show the obvious tradeoff between the quality of answers and the number of generated keys (and, therefore, the amount of the indexing traffic) in the network. Finally, we showed that real savings in storage and bandwidth requirements can be achieved with a marginal degradation of the answering quality.

## 6. RELATED WORK

A number of solutions for text-based retrieval in decentralized environments have been proposed in the literature. They are based on either unstructured [4], hierarchical [11, 2], or structured P2P networks [3] populated with peer-level collection descriptions to facilitate the peer-selection process followed by document-level retrieval at the selected peers. Such solutions depend on the quality of peer-level descriptors and in general perform well for clustered content when a small subset of peers holds documents relevant to a given query. On the contrary, our solution assumes a random distribution of documents over the peers and performs document-level indexing in structured P2P networks. It is the continuation of our efforts to design scalable solutions for full-text P2P search [13, 14, 18] with a goal of minimizing the traffic generated during indexing and retrieval.

While implementing full-text retrieval with structured P2P networks, the problem of long posting lists has been identified as a major obstacle [9]. When large posting lists are the major concern for global single-term indexing, the authors of [16] and [19] propose top-$k$ posting list joins and Bloom filters as promising techniques to reduce the search cost. However, the study reported in [23] shows that such optimizations cannot offer scalable retrieval for web-size document collections.

Similarly to our HDK approach, the KSS System [6] precomputes and stores results of inverted list intersections for popular queries. However, exhaustive term combinations generation leads to unrealistic storage requirements for the index. An extension of the KSS system is presented in [10]. This technique requires a query log available in advance and might not adapt well to future queries. Our approach is different to KSS since the index is constantly evolving by activating new keys during the query processing.

The importance of term co-occurrences has recently been identified in [12] where profitable term combinations are associated with corresponding "peerlists", thus improving the peer selection process during querying. While performing well in a small network it is unclear whether the retrieval quality will remain acceptable in a large-scale setting due to the peer granularity of the index used in this approach.

The work presented in [20] uses a hybrid indexing structure to reduce the bandwidth consumption during the querying process. In such a hybrid index, posting lists contain not only document references, but also top-k terms of a document. This index facilitates local multi-term query processing at the peer responsible for any query term. This approach requires an overhead in storage space and the query expansion process must be applied to guarantee a satisfactory retrieval quality. Notice, that a similar idea was used in the DCT approach [17] to store caches.

## 7. CONCLUSION

In this paper we presented a novel query-driven indexing strategy for multi-term query processing in structured P2P networks. Our QDI approach avoids maintenance of rarely used index entries by adapting to the query popularity distribution observed in the query log. In summary, the paper makes the following contributions in the P2P information retrieval area: 1) introduces a P2P query-driven indexing paradigm, 2) outlines the design of the novel QDI approach, based on popular and highly-discriminative keys, 3) shows the scalability analysis based on both theoretical and experimental evaluations, 4) provides an estimate of the QDI approach retrieval performance with real web-size document collections and the Wikipedia query logs.

Whereas a purely data-driven approach would lead to generation of a large number of keys, generation of keys from a substantially smaller query log facilitates more compact and targeted indexing structure. Thus, the bandwidth consumption while indexing is significantly reduced as no additional traffic is used for the maintenance of superfluous index entries. At the same time query processing generates bounded traffic thanks to the limited posting list size. Moreover, by adjusting the minimum popularity of candidate term combinations found in the query log, we can tradeoff the size of the index and the bandwidth consumption with the query answering quality.

## 8. REFERENCES

[1] K. Albrecht, R. Arnold, M. Gahwiler, and R. Wattenhofer. Join and Leave in Peer-to-Peer Systems: The Steady State Statistics Service Approach. Technical Report 411, ETH Zurich, 2003.

[2] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. DL Meets P2P - Distributed Document Retrieval Based on Classification and Content. In *ECDL*, 2005.

[3] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Improving Collection Selection with Overlap Awareness in P2P Search Engines. In *SIGIR*, 2005.

[4] 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. In *HPDC*, 2003.

[5] A. Datta, M. Hauswirth, R. Schmidt, R. John, and K. Aberer. Range Queries in Trie-Structured Overlays. In *P2P*, 2005.

[6] O. D. Gnawali. A Keyword Set Search System for Peer-to-Peer Networks, 2002. Master's thesis, MIT.

[7] H. S. Heaps. *Information Retrieval: Computational and Theoretical Aspects*. Academic Press, Inc., 1978.

[8] F. Klemm, J.-Y. L. Boudec, and K. Aberer. Congestion Control for Distributed Hash Tables. In *NCA*, 2006.

[9] J. Li, B. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris. The Feasibility of Peer-to-Peer Web Indexing and Search. In *Workshop on Peer-to-Peer Systems*, 2003.

[10] Z. Liang, Z. Fu-tai, and M. Fan-yuan. KRBKSS – a Keyword Relationship Based Keyword-Set Search System. *SCIENCE*, 2005.

[11] J. Lu and J. Callan. Federated Search of Text-Based Digital Libraries in Hierarchical Peer-to-Peer Networks. In *ECIR*, 2005.

[12] S. Michel, M. Bender, N. Ntarmos, P. Triantafillou, G. Weikum, and C. Zimmer. Discovering and Exploiting Keyword and Attribute-Value Co-occurrences to Improve P2P Routing Indices. In *CIKM*, 2006.

[13] I. Podnar, M. Rajman, T. Luu, F. Klemm, and K. Aberer. Beyond Term Indexing: A P2P Framework for Web Information Retrieval. *Informatica, Special Issue on Specialised Web Search*, 2006.

[14] I. Podnar, M. Rajman, T. Luu, F. Klemm, and K. Aberer. Scalable Peer-to-Peer Web Retrieval with Highly Discriminative Keys. In *ICDE*, 2007.

[15] W. J. Reed. The Pareto, Zipf and other Power Laws. *Economics Letters*, 74(15-19), 2001.

[16] P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. In *Middleware*, 2003.

[17] G. Skobeltsyn and K. Aberer. Distributed Cache Table: Efficient Query-Driven Processing of Multi-Term Queries in P2P Networks. In *P2PIR*, 2006.

[18] G. Skobeltsyn, T. Luu, I. Podnar Žarko, M. Rajman, and K. Aberer. Query-Driven Indexing for Peer-to-Peer Text Retrieval. In *WWW*, 2007.

[19] T. Suel, C. Mathur, J.-W. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram. ODISSEA: A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval. In *WebDB*, 2003.

[20] C. Tang and S. Dwarkadas. Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval. In *NSDI*, 2004.

[21] http://ir.dcs.gla.ac.uk/terrier/.

[22] http://en.wikipedia.org.

[23] J. Zhang and T. Suel. Efficient Query Evaluation on Large Textual Collections in a Peer-to-Peer Environment. In *P2P*, 2005.