PCIR: Combining DHTs and Peer Clusters for Efficient Full-text P2P Indexing

Odysseas Papapetrou, Wolf Siberski, Wolfgang Nejdl

L3S Research Center, Leibniz Universität Hannover, Germany

Abstract

Distributed hash tables (DHTs) are very efficient for querying based on key lookups. However, building huge term indexes, as required for IR-style keyword search, poses a scalability challenge for plain DHTs. Due to the large sizes of document term vocabularies, peers joining the network cause huge amounts of key inserts and, consequently, a large number of index maintenance messages. Thus, the key to exploiting DHTs for distributed information retrieval is to reduce index maintenance costs. Various approaches in this direction have been pursued, including the use of hybrid infrastructures, or changing the granularity of the inverted index to peer level. We show that indexing costs can be significantly reduced further by letting peers form groups in a self-organized fashion. Instead of each individual peer submitting index information separately, all peers of a group cooperate to publish the index updates to the DHT in batches. Our evaluation shows that this approach reduces index maintenance cost by an order of magnitude, while still keeping a complete and correct term index for query processing.

1. Introduction

Distributed hash tables (DHTs) are the basis for most modern Peer-to-peer (P2P) Information Retrieval (IR) systems. They are preferred over super peer or other hierarchical network structures for their robustness, efficiency, and scalability. They also outperform unstructured networks in query execution efficiency and in information retrieval quality.

As such, DHTs provide the necessary overlay when it comes to building P2P applications with IR features. For example, SCOPE [1, 2] is a P2P social networking system which uses DHT for scalability. SPROUT enhances the search functionality in P2P networks by maintaining a social network on top of a standard DHT. UniWiki [3] employs a DHT overlay to build a distributed, fault-tolerant and scalable Wiki system. ALVIS [4] builds an inverted index over a P2P network, which enables scalable cooperative search, and Minerva [5, 6] facilitates a DHT overlay to allow web search.

These DHT-based systems maintain a distributed inverted index which maps each term/key to the list of relevant peers or documents. Each peer joining the network also joins the DHT infrastructure, and publishes its contents at the distributed inverted index. For query processing, the query terms are looked up in this index and all relevant documents or peers are found. This approach (in the following called *flat DHT indexing*) works well when the number of terms per peer is limited, even for extremely large networks.

However, a common issue of systems using flat DHT indexing is that maintaining the DHT inverted index becomes prohibitively expensive when the number of distinct terms per peer, is high [7]. When each peer features a large amount of distinct terms to index, it needs to execute a large amount of DHT lookups for indexing its collection. And since the DHT decides which peer is responsible for which term via hashing, a considerable number of peers holding some part of the DHT have to be contacted to fully publish all terms of a peer. High peer churn and frequent document updates in the peers aggravate the problem.

Several approaches have been proposed to limit index maintenance cost, e.g., combining DHTs with unstructured networks [8], or indexing only selected terms [9]. While these approaches efficiently reduce index maintenance costs, they sacrifice completeness of the inverted index, increase the cost of query execution, or decrease the quality and completeness of the query results (see Section 7 for a comprehensive overview).

In this work, we present a novel approach which reduces indexing costs significantly without affecting querying cost or result quality. The approach is based on a hybrid super peer/DHT topology: we organize peers into groups, each of them represented by a super peer for publishing the group's information to a single global DHT used for query processing. Each peer independently joins a group, and submits its index to the representative super peer for the group. The group representative efficiently batches this information and periodically publishes it to the DHT, yet without removing any details or compromising the completeness of the inverted index. Because of this message batching, the number of the required DHT lookups for publishing and the total number of messages is reduced. Our evaluation shows that the proposed approach enables cost savings of up to one order of magnitude compared to a plain DHT, regarding number of messages as well as total transfer volume. At the same time, network workload of the super peers remains below the network workload of the regular peers in the flat DHT approach.

In its basic form, our algorithm relies on arbitrary assignment of peers to groups, without taking the peer contents into

Email addresses: papapetrou@l3s.de (Odysseas Papapetrou), siberski@l3s.de (Wolf Siberski), nejdl@l3s.de (Wolfgang Nejdl)

account. We achieve further improvements by introducing a distributed clustering scheme, such that peers form groups/clusters based on content similarity. This reduces the total number of distinct terms per cluster and additionally decreases the required DHT lookups for publishing the terms and the total number of messages for maintaining the DHT. The algorithm presented here is based on preliminary work published in [10].

Contributions The main contributions of this work are:

- We propose a two-level P2P full-text indexing approach, which reduces full-text indexing costs significantly. Our algorithm maintains a complete inverted index, and does not affect the querying performance and IR efficiency.
- We propose an algorithm for distributed clustering in P2P systems. The algorithm clusters peers that share many common terms together. Unlike previous P2P clustering approaches, our approach exhibits low costs even for high-dimensional data like text and scales to large networks.
- We combine distributed clustering with the two-layered full-text indexing infrastructure, for further reducing the indexing costs. Compared to previous full-text indexing approaches, our clustering-enhanced approach reduces the indexing costs by an order of magnitude for large P2P networks.

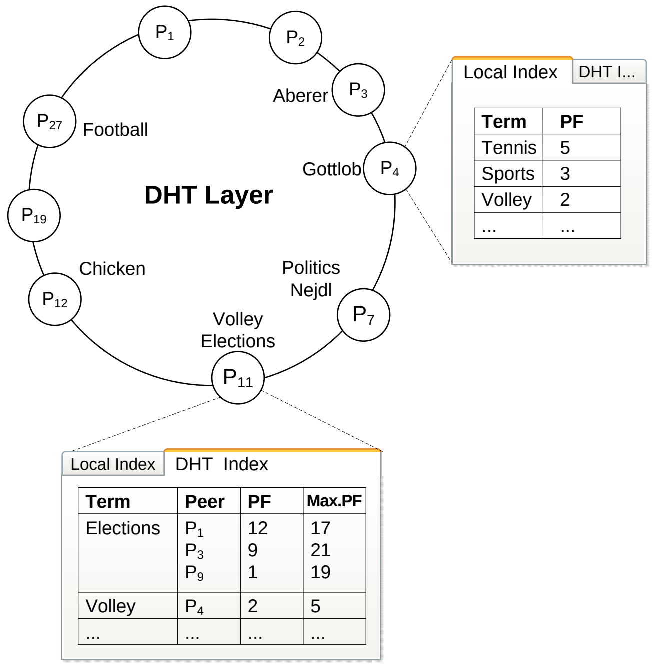
The paper is structured as follows. The next section shows how inverted indexes are currently constructed over DHTs and how the resulting index structure is used to evaluate keyword queries. We present the basic algorithm and the building blocks of our topology in Section 3. In Section 4 we introduce our distributed clustering scheme and show how it enhances the basic algorithm, reducing the network overhead of the inverted index maintenance process. A cost analysis is presented in Section 5, and Section 6 contains the results of our experimental evaluation. We review related work in Section 7. We conclude with discussion and future work.

2. Background

This work is focused on efficient creation and maintenance of an inverted term index in a DHT. In this section we describe the underlying techniques and the standard approach for DHT-based information retrieval in P2P networks.

2.1. DHT-based Inverted Indices

Distributed Hash Table networks (DHTs) provide efficient hash table capabilities in a P2P environment by arranging peers in the network according to a specific graph structure, usually a hypercube. DHTs offer the same functionality as their centralized hash table counterparts, a $put(key, value)$ method, which associates value with key, and a $get(key)$ method, which returns the value associated with key. Both methods have a cost of $O(\log(n))$ messages, where n is the number of participating peers. In the following, we use Chord [11] as underlying system, but our approach does not rely on a specific DHT approach.



Term	PF
Tennis	5
Sports	3
Volley	2
...	...

Term	Peer	PF	Max.PF
Elections	P_1	12	17
	P_3	9	21
	P_9	1	19
Volley	P_4	2	5
...	...	...	...

Figure 1: Flat DHT topology and index structure

Each peer participates as a node in the DHT, by taking responsibility for part of the key range. Peers also index their collections in the DHT inverted index. For the latter, each peer first builds a local inverted index for its documents, after processing them with stopword filtering and stemming as usual (see Peer P_4 in Figure 1 as example). Then, it creates an index entry for each discovered term and adds it to the DHT by sending it to the peer responsible for this term. Terms are mapped to peers in the DHT by using a hash function. The distributed inverted index can be built on any DHT implementation, since it only requires the generic insert/lookup functionality, offered by all DHT overlays.

All common scoring functions in information retrieval take into account the term frequency $tf(t, d)$, the number of occurrences of term t in document d. Some P2P information retrieval systems build and maintain such a document-granularity inverted index, e.g., [4]. However, indexing each document individually is expensive in terms of network cost. Therefore, most P2P IR approaches use peer granularity [5], i.e., instead of publishing document term frequencies, each peer publishes its peer term frequency $pf(t, p_i) = \sum_{d \in DC(p_i)} tf(t, d)$, where $DC(p)$ is the set of documents held by peer p_i. Peer-granularity indexes are more compact and can be efficiently exchanged in a distributed system. Thus, they provide a good balance between precision and resource requirements [12]. See Section 7 for a detailed description of all indexing variants.

For the identification of relevant peers, the peer frequencies need to be normalized by the size of each peer's collection. Therefore, in addition to the peer frequency, the maximum peer frequency $max\_pf(p_i) = \max_{t \in DC(p_i)} pf(t, p_i)$ is stored in an index entry. We discuss how peer frequency and maximum peer frequency are used for query processing in Section 2.2.

In the following we assume that an index entry for term t and peer p_i contains the contact information for p_i (usually the IP address), the peer frequency $pf(t, p_i)$, and the maximum peer frequency $max\_pf(p_i)$ (see Figure 1). The list of all index entries published for a term is called a *posting list*. The posting list of a term t can be retrieved by performing a DHT lookup, using t as a key.

We refer to the described approach as *Flat DHT Indexing*, as it does not use an intermediate layer between the peers and the DHT to optimize index maintenance. Flat DHT Indexing is the standard approach for index maintenance employed by recent P2P systems, and we use it as the baseline for evaluating the performance of our approach.

In a P2P network, peers might become disconnected unexpectedly. The usual approach to cope with this churn is to attach an expiration time to each index entry. Peers periodically republish their content such that their index entries get refreshed before they expire. Under the periodic republishing model, it is acceptable for the distributed index to have stale data of maximum age *period*, if the DHT republishing occurs with an interval *period′*, with $period \leq period'$. The periodic republishing model is favorable for realistic high-churn P2P setups, and the interval length *period′* can be configured to balance the index accuracy with the overall publishing cost.

While DHT-based Inverted Indices form an excellent foundation for distributed query processing, their creation and maintenance incurs a very high number of network messages. Even if these messages are usually small, their packaging according to the network protocol (i.e., TCP/IP frames) causes a high network volume overhead. For the TCP/IPv4 protocol, this overhead is 75% of the total message size, considering the theoretic minimum TCP/IP frame size of 32 bytes (64 bytes for a transaction) and assuming an average of 10 bytes for each term. The large number of these messages also causes a high workload to the intermediate network routers. Furthermore, the average cost at the individual peers which are asked to submit or route these messages can render the weaker peers unable to participate, and cause increased latencies even at the high-end peers.

2.2. DHT-based Query Processing

The work described here focuses on efficiently maintaining the required information in the distributed inverted index, and it is applicable to all query execution techniques that require a DHT-based index. For completeness, we summarize one of these techniques here, where query processing is performed as two-step process [5]. In this approach the query initiator first discovers all the related peers and selects the most relevant of them (collection selection). It then queries the selected peers, collects and merges the results, and presents them to the user (query execution).

The described approach employs a peer-granularity index. However, PCIR can also efficiently be used to optimize document-granularity indexing [4] as well as query-driven indexing [13, 14], as we show in Section 6.

Collection selection. There is rich literature on collection selection techniques for distributed collections. The work is directly applicable to P2P systems, since each peer can be considered as a distributed collection. In this work we use CORI [12] to identify relevant peers

Assume a query Q consisting of terms $\{t_1, t_2 \dots t_l\}$. The query initiator looks up each term in the DHT, and retrieves the posting lists for all query terms. These posting lists contain the peer scores per term for all related peers. The CORI score $s(Q, p_i)$ for query Q and peer p_i is computed as:

$$s(Q, p_i) = \frac{1}{|Q|} * \sum_{\forall t \in Q} d_b + (1 - d_b) * T_{t,p_i} * I_{t,p_i}$$

with $T_{t,p_i} = d_t + (1 - d_t) * \frac{\log(pf(t,p_i)+0.5)}{\log(max\_pf(p_i+1.0))}$ and $I_{t,p_i} = \frac{\log(\frac{n+0.5}{cf(t)})}{\log(n+1.0)}$. Parameters d_t and d_b are constants with a default value of 0.4 (for more information on how to set these constants see [12]). $pf(t, p_i)$ is the frequency of term t in peer p_i and $cf(t) = |\{p | pf(t, p) > 0\}|$ is the collection frequency for t, that is, the number of peers at which term t occurs. The frequency of the most frequent term in peer i is denoted as $max\_pf(p_i)$. The parameter n is the number of peers connected in the network.

Comparing the data stored in the DHT (Figure 1) and the data required for CORI score function, we see that all the required data is already stored in the DHT, except for n, the total number of peers. The value of n can be inexpensively estimated with a random walk [15], and periodically redetermined to take churn into account. Thus, for each query term the necessary parameters can be easily retrieved by only one DHT lookup per term.

Query execution. The query initiator finds the α peers with the highest CORI scores for the query, and routes the query to them. The respective inverse collection frequency $icf(t) = \frac{1}{cf(t)}$ for each query term t is attached to the query, to allow all peers to properly weight the importance of each term. Each peer then independently selects its top-k related documents, using $tf * icf$ score. Links to the results are returned to the query initiator and they are ordered by relevance with respect to the query.

2.3. Bloom Filters

In this work we use Bloom filters for compactly representing large term sets (Section 4). Bloom filters were first proposed in [16], as a space-efficient representation of sets $S = \{e_1, e_2, e_3 \dots e_n\}$ of n elements from a universe U. A Bloom filter consists of an array of m bits and a set of k independent hash functions $F = \{f_1, f_2, \dots, f_k\}$, which hash elements of U to an integer in the range of $[1, m]$. The m bits are initially set to 0 in an empty Bloom filter[1]. An element e is inserted into the Bloom filter by setting all positions $f_i(e)$ of the bit array to 1, for all $f_i \in F$. Bloom filters are used as content summaries in our algorithm.

A limitation of Bloom filters is that they do not allow removing of elements. For removing an element from a Bloom filter, the whole Bloom filter needs to be rebuilt from scratch. A workaround was proposed by Fan et al. [17], called *counting*

[1]We use the expressions ‘A bit is set to true/false’ and ‘A bit is set to 1/0’ interchangeably.

Bloom filters. A counting Bloom filter replaces the bit array of standard Bloom filters with an array of *m* counters. For adding an element in a counting Bloom filter, the element is hashed using the hash functions, and all respective counters are increased by one. For removing an element, respective counters are decreased by one. In this work, counting Bloom filters are used to compute the difference between two Bloom filter summaries.

3. PCIR Basic Algorithm

As explained in the previous section, the main issue with DHT-based inverted indexes is their high maintenance cost. PCIR reduces this cost without changing the content of the resulting index, so that existing query execution techniques can be applied, like the one described in Section 2. Since PCIR creates exactly the same inverted index as flat DHT indexing, query processing does not need to be adapted, and the same retrieval quality as before is achieved.

Our approach builds on the observation that peers usually have a large term overlap. In the flat DHT model, this term overlap is not exploited; each peer independently publishes its terms in the DHT, and therefore it requires its own DHT lookups for all its terms. PCIR, short for *Peer Clusters for Information Retrieval*, exploits this term overlap between peers by forming small groups of peers, and assigning responsibility for DHT lookups in each group to a selected super peer of the group.

Figure 2 illustrates the PCIR overlay. First, a new peer joins the DHT, but without publishing its terms in the DHT inverted index. Second, it either discovers and joins an existing group of peers, or creates a new group and assumes the role of the super peer. Third, it builds the local inverted index for its collection and sends it to the super peer of its group. In turn, the super peer of each group collects these inverted indices of the group's peers and publishes them to the DHT. Super peers reduce the number of DHT lookups by performing only one lookup per term in the group, regardless of the number of collected entries for the term. The above steps are repeated periodically to compensate churn.

It is important to note that the super peers do not post an aggregated inverted index for the group, they post the original index entries – the posting lists – as received from their group peers. Therefore, unlike other super peer IR networks, queries do not go through the super peers, and query processing does not impose additional workload on the super peers. Super peers in PCIR only contribute to the indexing process but do not need to act as a point of entry for queries.

Distributed Inverted Index. For the purpose of query execution we construct a global inverted index over a DHT. For demonstration purposes we now use peer granularity inverted indices, as described in Section 2, although as we show in Section 6, PCIR is also beneficial to other publishing strategies, e.g., document-granularity indexing. The distributed index created from PCIR looks nearly identical as the one constructed by flat DHT Indexing (cf. Section 2). The only difference is that in addition to the peer information, the super peer that inserts the entry in the term's posting list, also adds its own contact information. As we explain later, this information is required for efficiently building the peer groups. Figure 3 shows a sample index.

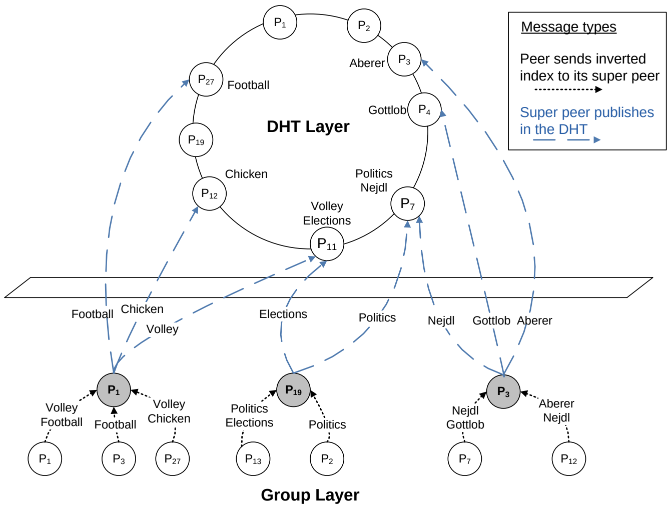


Figure 2: The two-layer architecture combines grouping of peers around super peers and an inverted index over a DHT (super peers are gray shaded)

Keyword	Peer	*pf*	*max_pf*	Super peer
Football	p_1	12	17	p_4
	p_3	9	21	p_4
	p_9	1	19	p_9
Volley	p_1	7	17	p_4
...	...	...	...	...

Figure 3: Logical DHT Inverted Index. For each term, the DHT keeps a list of relevant peers, their contact details and the respective peer scores and super peers.

Construction of the inverted index relies only on the generic DHT insert/lookup functionality. Therefore we abstract from the implementation details of DHT. Although in this work we use Chord [11] as DHT infrastructure, any DHT implementation can be used. In the following, we explain in detail how this inverted index is built and maintained.

3.1. Peer Life-cycle

Having described the structure of the DHT inverted index built by our algorithm, we now take a closer look at the life-cycle of the peers in the network.

Peer joining the network. This activity includes two tasks: (a) joining the DHT, and (b) joining a peer group. We omit the former from the discussion since it is specific to the DHT protocol and orthogonal to our work.

Alg. 1 presents the process for joining a group. The joining peer first decides whether it should create its own peer group and become a super peer, or join an existing group as a normal peer. In the former case, it publishes its contents to the DHT directly, and awaits other peers to join its peer group. In the latter case, the peer needs to find a super peer that still accepts connections. It does so by running a DHT lookup on a random value (in the case of Chord, which typically has an id ring from 0 to 2^{160}, the peer chooses a random value in this range). The DHT lookup returns a peer responsible for holding the respective hash value, which is in turn queried for all the super peers it is aware of (i.e., the super peers that have published information in the posting lists this peer holds). The retrieved super peers are then checked in random order. The new peer joins the peer group of the first discovered super peer that can accept it, i.e., it is not overloaded.

The decision for becoming a super peer or a normal peer can be taken by each peer independently. For instance, the desired ratio *sp_ratio* of peers and super peers can be used to determine the probability of each individual peer to become a super peer. Another strategy for a peer would be to become super peer after a given number of unsuccessful attempts to join an existing peer group. Unlike previous super peer systems (e.g., [18]), PCIR does not require super peers to be especially powerful. A super peer in PCIR typically has less network workload than a regular peer in flat DHT indexing.

PCIR super peers can also independently control their workload and stop accepting new peers before they become overloaded. Each super peer sets its own *upper bound* for its workload, based on its available network and computational resources. Since workload of super peers depends on the number of distinct terms in the group, in our implementation super peers express their upper bound in maximum number of distinct terms in their group. Other possible ways of expressing the upper bound are maximum number of peers in the group, maximum group collection size, or a combination of the above. Note that this limit will never cause index entries to get lost. As soon as a super peer reaches its self-imposed limit, it just stops accepting new peers.

Super peers do not have a special role during query processing; they are only used to make the DHT publishing more efficient. Therefore, their attributes, performance and workload, do not affect querying performance or quality of query results. A long uptime of super peers is also not required, since the DHT update protocol is based on periodic republishing; when a super peer disconnects, the peers in its group simply repeat the process and join other groups.

Algorithm 1 Peer joining a peer group (basic PCIR)

```
1:  joinPCIR() {
2:  if decideIfSuperPeer() then
3:     // become a super peer
4:     this.isSuperPeer:=true;
5:     groupInvertedIndex:=myInvertedIndex;
6:  else
7:     // become a normal peer
8:     this.isSuperPeer:=false;
9:     // find random super peer
10:    mySuperPeer:=findRandomSuperPeer();
11:    // and submit your inverted index
12:    mySuperPeer.groupInvertedIndex.update(myInvertedIndex);
13: end if
14: }

15: SuperPeer findRandomSuperPeer() {
16: repeat
17:    int randomKey:=getRandom();
18:    peer p:=DHTLookup(randomKey);
19:    <SuperPeer> listOfSPs:=p.allKnownSuperPeers();
20:    while (listOfSPs.isNotEmpty() & mySuperPeer is null) do
21:       SuperPeer randomSP:=listOfSPs.selectRandom();
22:       if randomSP.canAcceptMe() then
23:          mySuperPeer:=randomSP;
24:       end if
25:    end while
26: until SuperPeer found
27: return mySuperPeer;
28: }

29: boolean decideIfSuperPeer() {
30: RandomNumber random:=getRandom(0,1);
31: return (random ≤ $P_{sp\_ratio}$);
32: }
```

Peer/Super peer periodic publishing. After a peer joins a group it periodically sends its inverted index to the super peer of that group. The super peer of each group packs together the peer frequencies per term (the posting lists for its group), and publishes them in the DHT (Alg. 2). The publishing process is periodically repeated to compensate churn.

This two-layered publishing process has several advantages compared to flat DHT indexing. The peers can efficiently publish their inverted index at their super peers; publishing requires only one message which is easily compressible, and does not generate any DHT lookups. The super peers can also optimize the updating of the DHT index. All the group's peer scores for each term are packed in a single message, thereby (a) requiring only a single DHT lookup and only one publishing message per distinct group term, and, (b) enabling compression and delta updating. A disadvantage of the two-layered approach is that the data now needs to be published twice, the first time from each peer to its super peer, and the second time from the super peer to the DHT inverted index. However, the benefits of the two-layered architecture, and mainly the drastic reduction of the DHT lookups, surpass the extra publishing overhead.

Handling Churn. PCIR relies on periodic publishing of the peer scores, both from peers to super peers, and from super peers to the DHT. Particularly, when a peer sends its inverted index to its super peer, it attaches an expiration period. The super peer uses this expiration period to keep the group's inverted index updated, i.e., to remove the expired entries from the group's posting lists. Therefore peers do not need to unpublish old information; these are automatically filtered out by the super peers. Similarly, super peers attach an expiration period to each of their DHT publishings. The peers participating in the DHT index detect and remove the expired posts, so super peers also do not need to unpublish anything. A global system time is not required. All peers only need to share the same expiration period and no further synchronization between the peers is required.

Algorithm 2 Periodic Peer/Super Peer publishing

```
while true do
    if this.isSuperPeer then
        // update with my own collection
        this.groupInvertedIndex.update(myInvertedIndex);
        // publish group's inverted index
        for all term t in groupInvertedIndex do
            // publish peerlist in DHT
            peer p:=DHTLookup(t.hashValue);
            p.publish(groupInvertedIndex.getPeerList(t));
        end for
    else
        if !(mySuperPeer.isAlive()&mySuperPeer.canAcceptMe()) then
            joinPCIR(); // rejoin a group
        end if
        mySuperPeer.groupInvertedIndex.update(myInvertedIndex);
    end if
    sleep(period);
end while
```

Because of the periodic republishing, peers do not have to act in the case a regular peer leaves the network, either by expected departure or by unexpected failure. The DHT itself automatically recovers without loss of data (see for example [11, 19]). The peer's summary published at the super peer will also eventually expire, get removed from the super peer and, in turn, from the DHT inverted index. In the meantime, any query routed to this disconnected peer will simply fail, and the next relevant peer will be selected for querying.

Super peers may also disconnect from the network. In this case, the group's peers detect the failure of the super peer in the next publishing period and individually find and join another group, as described in Alg. 1. Due to super peer churn, the publishings of a peer might not be propagated in the DHT within one publishing period. The probability that this happens for i consecutive periods is small, and in particular, $pr = 1 - (1 - c)^i$. For example, for a churn of 20%, the probability that the contents of a peer are not published after 3 periods is less than 1%. To avoid loosing already published information from the DHT, we set the index entry expiration period to a multiple of the republishing period length, e.g., 3 times the republishing period. This ensures availability of the entries even under super peer churn, without requiring the peers to take specific actions when their super peer fails. Therefore, super peer churn does not impose additional cost on the peers.

Reducing the cost of super peers. For reducing the network usage, all peers use delta updating to send their updated inverted indices to their super peers. In case peers have no changes, they send a single *keep-alive* message to notify their super peers that they are still alive and they have no changes. These optimizations significantly reduce the network cost of super peers, as well as the total network cost. Note that these optimizations cannot be used in the flat DHT publishing scenario, as the DHT lookup and the *keep-alive* or delta message would still be required per distinct term per peer, and the total number and size of messages would not be reduced.

An additional measure taken towards reducing the cost of super peers involves the DHT lookup implementation. In this work we use recursive DHT lookups: lookups are handled recursively by the DHT peers, without requiring interaction with the originators of the lookup at each hop. Apart from the efficiency benefits of recursive compared to iterative DHT lookups [20], recursive DHT lookups distribute the lookup cost to all DHT peers more evenly. Super peers, which initialize the majority of the DHT lookups, only need to execute the first lookup hop. The remaining lookup hops are distributed evenly to all DHT peers.

4. PCIR Clustering-enhanced Algorithm

Basic PCIR algorithm reduces the required DHT lookups by exploiting the term overlap in the group's collection; only one DHT lookup is created for each distinct term in the peer group. This basic approach already accounts for a 5-times reduction of the total number of messages. The number of required messages can be further reduced by reorganizing peers in the groups so that peers with similar content end up in the same peer groups. As a result, super peers of the groups will have fewer distinct terms, and they will need to perform fewer DHT lookups for publishing the total terms of the collection, yet without compromising completeness of the inverted index.

The clustering-enhanced PCIR algorithm clusters the peers based on their contents, so that peers with similar contents are assigned to the same group. We refer to these groups as *peer clusters*. For building the peer clusters we propose an inexpensive clustering algorithm based on Bloom filter representations for the cluster and peer centroids. Each peer creates a Bloom filter representation of its centroid by hashing all its terms in an empty Bloom filter. Super peers are additionally responsible for maintaining the Bloom filter representation of their cluster's centroid, which equals to the disjunction of the Bloom filters of all the peers belonging to the cluster. For estimating the term overlap between a peer and a candidate cluster, the corresponding Bloom filter representations are compared.

To avoid comparing each peer's Bloom filter with the Bloom filters of all clusters, clustering-enhanced PCIR employs a DHT-based inverted index to index the clusters: Super peers index their top most frequent terms in the DHT. When a new peer joins the network, it first identifies its top most frequent terms, and uses the inverted index to discover all clusters with common top terms. As we show later, this process offers probabilistic guarantees that the peer joins the best cluster – the cluster with the higher term overlap.

In the following we present the building blocks of clustering-enhanced PCIR. In Section 4.1 we present the clustering objective function, and show how it is inexpensively estimated using the Bloom filter representations of the peer and cluster centroid. In Section 4.2 we describe a cluster centroid caching scheme which is used to alleviate the workload of super peers. Section 4.3 describes the changes at the DHT inverted index that enable efficient peer clustering. The process of joining peers in the clustering-enhanced PCIR network is put together in Sec-

λ	Number of top frequent terms published per super peer
μ	Maximum number of clusters to compare each virtual peer's Bloom filter with
BF_{P_i} / BF_{C_x}	Bloom filter of peer P_i / cluster C_x
m	Bloom filter length in bits
k	Number of hash functions per Bloom filter
$tb(BF_i)$	Number of bits set to true in BF_i

Table 1: Notations for clustering-enhanced PCIR

tion 4.4. In all other aspects, clustering-enhanced PCIR algorithm works like the basic algorithm.

4.1. Clustering objective function

Clustering is used for increasing the term overlap in super peers. The clustering objective function needs to compute the overlap between a peer and a cluster collection, and assign the peer to the cluster with the larger overlap. To avoid exchanging large inverted indices between peers for computing the overlap, peers estimate the cardinality of the overlap using Bloom filters [16].

The clustering objective function is formally defined as follows:

Definition Given a peer P_i, and the Bloom filter for its collection BF_{P_i}. For a candidate cluster C_x, with Bloom filter centroid BF_{C_x}, the objective function $f(P_i, C_x)$ is:

$$f(P_i, C_x) = \begin{cases} f'(BF_{C_x}, BF_{P_i}) & \text{, if } P_i \notin C_x \\ f'(BF_{C_x - P_i}, BF_{P_i}) & \text{, if } P_i \in C_x \end{cases} \quad (1)$$

where $f'(BF_{C_x}, BF_{P_i})$ gives the expected cardinality for the overlap between the cluster C_x and the peer P_i using their Bloom filters (cf. Theorem 4.1), and $BF_{\{C_x - P_i\}}$ is the Bloom filter of cluster C_x after removing the contents of the peer P_i. The best cluster is the one that maximizes the objective function.

The following theorem shows how to estimate the cardinality of the overlap between a cluster and a peer using Bloom filters.

Theorem 4.1. *Let BF_A and BF_B denote the Bloom filters of collections A and B respectively. With $BF_\wedge$ we denote the Bloom filter produced by bit-wise AND merging of the bit arrays of BF_A and BF_B. We assume that all Bloom filters are of the same size m, and use the same k hash functions. Then the expected number of elements in $A \cap B$ is:*

$$E(|A \cap B|) = \frac{m^2 - m * (tb(BF_A) + tb(BF_B)) + tb(BF_A) * tb(BF_B)}{k * \ln(1 - 1/m)} - \frac{m^2 - m * (tb(BF_A) + tb(BF_B) - tb(BF_\wedge))}{k * \ln(1 - 1/m)} \quad (2)$$

where $tb(BF_x)$ is the number of bits set to true in BF_x.

The proof is included in the appendix.

Correctness of the objective function. The goal of peer clustering is to increase the term overlap at super peers. The objective function is correct if, given a peer and a set of candidate clusters, it selects the cluster that maximizes term overlap with the peer. Since Bloom filters are probabilistic, it may happen that the objective function picks the wrong cluster as the optimal. Although a wrong peer clustering decision does not affect the quality in terms of information retrieval, we compute the probability that the clustering objective function chooses the wrong cluster, to show that moderate Bloom filter sizes are enough for high-quality clustering.

Theorem 4.2. *Let BF_{P_i}, BF_{C_x} and BF_{C_y} represent the Bloom filters of peer P_i and clusters C_x and C_y respectively. Wlog. assume that $f(P_i, C_x) > f(P_i, C_y)$, where $f(P_i, C_j)$ denotes the clustering objective function on the peer P_i and cluster C_j (Eqn. 1). Then, the probability of $|P_i \cap C_x| > |P_i \cap C_y|$ is at least:*

$$Pr\Big[|P_i \cap C_x| > |P_i \cap C_y|\Big] > 1 - exp\left(-f(P_i, C_y) * \left(\frac{2 * f(P_i, C_x) - 2 * f(P_i, C_y)}{2 * f(P_i, C_y) + f(P_i, C_x)}\right)^2 /4\right) * exp\left(-f(P_i, C_x) * \left(\frac{f(P_i, C_x) - f(P_i, C_y)}{2 * f(P_i, C_y) + f(P_i, C_x)}\right)^2 /2\right) \quad (3)$$

The proof is included in the appendix.

The important observation from Theorem 4.2 is that correctness probability increases exponentially with the value of $f(P_i, C_x) - f(P_i, C_y)$. Thus, even for small differences, the objective function is able to distinguish the right cluster with high probability.

4.2. Cluster centroid caching

Super peers are responsible for maintaining the Bloom filter for their cluster centroid. To avoid rebuilding the filter from scratch every time a peer joins or leaves a cluster, super peers maintain it using a counting Bloom filter. When a peer joins a cluster, it sends its inverted index to the super peer of the cluster. From the inverted index, the super peer generates the corresponding Bloom filter, and adds it to the cluster's counting Bloom filter. When the peer publishing expires, i.e., it is not updated in time, the super peer subtracts the peer's Bloom filter from the counting Bloom filter.

Super peers reduce their workload (both network and computational) by caching the Bloom filter of their cluster to one or more other peers in the network. We refer to these peers as *cache peers*. Super peers also register contact details for the cache peers in the DHT inverted index (see Fig. 4). When a peer P_i wants to compare its centroid with a cluster, it retrieves the cache location of the cluster centroid from the DHT, and forwards its centroid filter to the cache peer. The cache peer compares the two filters and returns the estimated overlap size to peer P_i. The additional network workload required for maintaining a fresh copy in the cache peer is negligible. In fact, we can show that the proposed caching is always beneficial for super peers, for reducing both their network and computational

Keyword	Peer	PF	Max PF	Super peer	Cache
Football	P_1	12	17	P_4	null
	P_3	9	21	P_4	null
	P_9	1	19	P_9	null
	null	7	21	P_4	P_3
Volley	P_1	7	17	P_4	null
...	...	...	...	...	...

Figure 4: Logical DHT Inverted Index for clustering-enhanced PCIR. Changes compared to the basic PCIR Index are gray-shaded.

workload, and that the workload of the cache holders is always less than the workload of the super peers.

To reduce network usage, super peers do not send counting Bloom filters in full resolution to cache peers. A counting Bloom filter at the super peer has 8 bits per counter, which allows a maximum of 256 values per counter. To execute the objective function (Eqn. 1), a cache peer only requires a counting filter of two bits per counter for finding both BF_{C_x} and $BF_{\{C_x-P_i\}}$; it only needs to know whether a counter at the counting filter of the cluster centroid has a value of 0, 1 or > 1. Thus, super peers reduce the counting Bloom filters to 2-bit per counter, which they send to cache peers. Cache peers can then compute BF_{C_x} and $BF_{\{C_x-P_i\}}$ without requesting more information from their super peers.

4.3. Aggregated Cluster Information Publishing

We enhance the DHT structure to enable efficient peer clustering. Apart from peer granularity information, each super peer additionally publishes aggregated cluster information for its cluster. For each of the cluster's top-λ most frequent terms, the super peer publishes an extra record to the DHT, which includes the overall cluster frequency and contact details of the peer holding the cluster centroid Bloom filter (the cache peer). The cluster granularity records are distinguishable from the peer granularity records, from their values in the *Peer* and *Cache* columns (see for example Fig. 4).

Publishing of the cluster-granularity data requires no additional messages; all cluster scores are piggy-backed on existing DHT-publishing messages. The network overhead for publishing each cluster score is only 24 bytes. Since only the top-λ highest cluster scores are published per super peer, with λ typically less than 10, the total additional network usage per super peer is usually less than 240 bytes per cluster.

4.4. Joining Peers in the Clustering-Enhanced Algorithm

As in basic PCIR, peers in clustering-enhanced PCIR first join the DHT and then find a super peer to attach to. The two algorithms differ in how peers select their super peers: peers in basic PCIR randomly select and join a super peer, whereas peers in clustering-enhanced PCIR select the super peer that maximizes the term overlap between the peer collection and the cluster collection.

Peers can evaluate the similarity of their collection and the candidate peer cluster's collections by using the clustering objective measure (Section 4.1). However, real-life peer collections, as real persons' interests, are often quite diverse with respect to the topics of interest. For example, a single peer may collect documents about the topics of *spontaneous nuclear fission*, *jazz music* and *Hollywood movies* all-together. Trying to find the best cluster for such multi-thematic peers is difficult, and may lead to suboptimal clustering.

We address this issue by partitioning each peer to a set of *virtual peers* with the use of document clustering. Ideally, each virtual peer focuses on a single subject, so that efficient clustering around super peers can be performed. Then, each virtual peer joins the best-matching cluster for its own collection, and posts its contents to the super peer of that cluster. While a peer splits its content into several virtual peers, it participates always as one node in the DHT. This approach reduces the average number of distinct terms per super peer, and therefore also reduces the overall maintenance cost for full-text indexing.

We use standard K-Means to cluster the documents and create the virtual peers. As clustering objective function, K-Means uses Jaccard similarity [21], which reduces the distinct terms per virtual peer. PCIR does not impose this clustering algorithm, though. Each peer is free to select the actual clustering algorithm, or even a partitioning hierarchy like MeSH, for determining its virtual peers. In general, better peer partitioning leads to better PCIR performance.

After a peer is partitioned to virtual peers, each of the virtual peers independently finds a suitable peer cluster to join. Searching for a suitable cluster is based on normal DHT lookups. First the virtual peer performs a DHT lookup for each of its top-λ most frequent terms and retrieves all the related clusters (Alg. 3, lines 7-16). For each related cluster, the virtual peer computes the partial cosine similarity based on the retrieved scores. Then (lines 23-31), for the μ clusters with the highest partial cosine similarity, it sends its Bloom filter to the peer that holds the cluster's centroid (the cache peer), and retrieves the expected overlap size (cf. Eqn. 1). Finally, it joins the most similar cluster based on the retrieved expected overlap sizes. If no suitable cluster is found, the virtual peer creates a new cluster and becomes the super peer (lines 19-21).

Execution of the described algorithm requires λ DHT lookups and at most μ peer-cluster comparisons. In total, finding and joining a cluster incurs a maximum of $\lambda * \log(n) + \mu$ total messages per virtual peer.

After a virtual peer joins a cluster, it periodically submits its inverted index to the super peer of that cluster. This requires only one message per virtual peer, and is effectively optimized by compression and delta updating. When a virtual peer sends its information, the respective super peer updates the local and cached copy of the cluster centroid to reflect all current information. Similar to the basic algorithm (Section 3), the only synchronization required between peers and super peers is a common expiration period; no global time is required.

Query processing and peer churn are handled in the same way as in the basic PCIR system (Section 3).

Algorithm 3 Peer periodically joining a peer cluster (clustering-enhanced PCIR)

```
// break to virtual peers
<virtualPeers>:=KMeans(myDocuments,numberOfClusters)
for virtualPeer vp ∈<virtualPeers> do
    vp.BF=ComputeMyBloomFilter(vp.localterms)
    Map<Cluster,Score> clusterScores;
    $TopTerms < T_1, T_2, \ldots T_\lambda >$:=TopK-Sort(vp.localterms, $\lambda$)
    for all term $t$ in $TopTerms$ do
        peer p:=DHTLookup(t.hashValue);
        <Cluster,Score>relevantClusters:=p.getClusterList(t);
        for all <c:Cluster,s:Score> in relevantClusters do
            if clusterScores.containsKey(c) then
                clusterScores(c):=clusterScores(c)+s;
            else
                clusterScores(c):=s;
            end if
        end for
    end for
    if clusterScores is empty then
        // become a super peer
        vp.isSuperPeer:=true;
        vp.clusterInvertedIndex:=myInvertedIndex;
    else
        Sort all clusters on their score desc.
        for top-$\mu$ clusters c do
            if c.canAcceptMe() then
                // the cluster is not overloaded
                cp:=CachePeer(c) // peer caching the centroid
                // compare my bf to cluster bf
                BFSimScore(c):=cp.compare(vp.BF,cp.clusterBF)
            end if
        end for
        Sort all clusters on the BFSimScore desc
        Join the cluster with the maximum BFSimScore
    end if
end for
```

5. Cost analysis

We now describe the cost model for the PCIR approaches. By cost, we refer to the number of required messages for the total system maintenance. The cost model does not include the messages for constructing and maintaining the Chord ring itself, as these are the same in all the approaches.

We compute the cost for the case that each peer rejoins the PCIR network at each iteration, i.e., it does not use delta updating, and at each iteration it needs to find the super peer from scratch, even if its old super peer is still available. This makes the cost analysis independent of the churn factor.

Throughout this section we use the following notations:

- n: Number of peers
- α: Average number of virtual peers per peer
- n_{sp}: Number of super peers
- D_p: Average number of distinct terms per peer
- D_g: Average number of distinct terms per group/cluster

5.1. Flat DHT Publishing

We first assess the cost for publishing all terms in a flat DHT setting. The expected cost (number of messages) for a DHT lookup [11] is: $C_{lookup} = \log(n)$. A peer requires on average D_p lookups, one for each distinct term, and an additional message per distinct term to publish the peer score in the DHT. Therefore, for a network of n peers the total cost of the system is:

$$C_{flat} = n * D_p * (\log(n) + 1) \tag{4}$$

5.2. Basic PCIR

The total cost in the basic approach is the sum of: (a) C_f : the cost for finding and joining a group, and (b) C_u : the cost for the super peers to update the DHT inverted index. We make the cost analysis churn-independent by assuming that each peer rejoins the network and finds a new super peer at each period. The following paragraphs provide the details.

(a) Finding and joining a group. Each peer requires one DHT lookup to find a super peer. Publishing all the data to the super peer requires one more message. The total number of messages C_f for all the peers to find and join a group is $C_f = n * (\log(n) + 1)$.

(b) Updating the DHT inverted index. Each super peer requires D_g DHT lookups, which cause a total of $D_g * \log(n)$ messages. In addition, publishing of the peer scores by the super peer requires D_g additional messages. Since we have n_{sp} super peers, the cost for all super peers to update the DHT is $C_u = n_{sp} * D_g * (\log(n) + 1)$ messages.

The total number of required messages for the basic approach is:

$$C_{basic} = C_f + C_u \leq n * \log(n) + n + n_{sp} * D_g * (\log(n) + 1) \tag{5}$$

5.3. Clustering-enhanced PCIR

Each peer partitions itself to α virtual peers, and each virtual peer behaves as a single peer. We therefore compute the cost of each virtual peer independently.

The total cost in the clustering-enhanced approach is the sum of: (a) C_f : the cost for the $\alpha * n$ virtual peers for finding and joining a cluster, and (b) C_u : the cost for the super peers to update the DHT inverted index. The following paragraphs provide the details.

(a) Finding and joining a cluster. For finding a cluster, a virtual peer performs a lookup on its λ most frequent terms in the DHT. This requires at most $\lambda * \log(n)$ messages per virtual peer. It then detects the top-μ most relevant clusters, sends its Bloom filter to the peers assigned the responsibility of caching the cluster Bloom filters, and retrieves the comparison result (the objective function values). This costs at most 2μ messages per virtual peer. Finally, each virtual peer submits its inverted index to the most similar super peer in a single message, and the super peer updates the cached copy of the Bloom filter (if required). The total number of required messages is upper-bounded by $C_f \leq \alpha * n(\lambda * \log(n) + 2 * \mu + 2)$.

(b) Updating the DHT inverted index. Each super peer needs to publish the inverted index for its cluster in the DHT. This requires $C_u = D_g * (\log(n) + 1)$ per super peer, like in the basic approach.

The total cost for the clustering-enhanced approach is upper bounded by:

$$\begin{aligned} C_{cluster} =& C_f + C_u \\ \leq& n * \alpha(\lambda * \log(n) + 2 * \mu + 2) + n_{sp} * D_g * (\log(n) + 1) \end{aligned} \tag{6}$$

5.4. Cost comparison

We now compare the expected cost for the basic PCIR and the flat DHT approach. For this we assume, as is common in IR [22, 23], that the dictionary size of each peer follows Heap's law [23]. We denote the length of a document, i.e., the number of words it consists of, as $len(d)$, and the length of a document collection $len(DC(p)) = \sum_{d \in DC(p)} len(d)$. According to Heap's law, the number of new terms added to the set of all terms by a new document decreases for each additional document. For the document collection $DC(p)$ of peer p, the number of distinct terms is $D_p \approx k * len(DC(p))^\beta$, where k and β are parameters dependent of language and text type.

Theorem 5.1. *Given a P2P network of n peers structured over a DHT, with dictionary size following Heap's law. Let C_{flat} denote the number of messages required by flat DHT indexing and C_{basic} the number of messages required by basic PCIR. The expected ratio of C_{basic}/C_{flat} is: $E(C_{basic}/C_{flat}) \approx \left(n_{sp}/n\right)^{1-\beta}$.*

The proof is included in the appendix.

From Theorem 5.1 we see that the expected ratio of C_{basic}/C_{flat} is dependent on: (a) the characteristic β value for the collection, and (b) the ratio of super peers to peers. Typical values for β are in the range of $0.4 < \beta < 0.6$, so the exponent is typically between 0.4 and 0.6. When the number of super peers decreases, the ratio gets higher, and the cost of the PCIR basic approach is reduced.

On the other hand, we do not want to overload the super peers. We can determine the number of required peers for a given average load, i.e., the number of terms $load_{SP}$ the super peer needs to publish. From Heap's law we can derive that $E\left(\frac{D_g}{D_p}\right) = (n/n_{sp})^\beta$ (see proof of Theorem 5.1 in the appendix). To get an expected number $load_{SP}$ of terms, we set $D_g = load_{SP}$. This gives us $load_{SP} = D_p * (n/n_{sp})^\beta$. By solving the equation for n_{sp}, we find the number of required super peers such that their average load is $load_{SP}$: $n_{sp} = n * (D_p/load_{SP})^{\frac{1}{\beta}}$.

Evaluation of the above equations requires knowledge of D_p and of the collection's characteristic β value. To estimate these values in real setups, we use sampling. Our experimental evaluation presented in Section 6 shows that sampling of a very small number of peers (0.3% of the total peers) is sufficient for estimating these values and thus for getting accurate cost estimations.

The expected number of messages per super peer can also be computed. Since each super peer needs to publish D_g terms, it requires $2D_g$ messages for initiating the DHT lookup and for publishing the terms. By participating in the DHT, a super peer is also required to route some DHT messages generated from other super peers while publishing their collections. The total number of these messages is $\approx n_{sp} * D_g * \log(n)$. Each super peer routes on average $n_{sp} * D_g * \log(n)/n$ of these messages. In addition, each super peer needs to receive the updates from all the peers belonging to its group, which cause an additional n/n_{sp} messages. The total number of messages per super peer is $2 * D_g + n_{sp} * D_g * \log(n)/n + n/n_{sp}$.

With respect to the clustering-enhanced approach, the cost ratio is:

$$\frac{C_{cluster}}{C_{flat}} \approx \frac{n_{sp} * D_g * (\log(n) + 1)}{n * D_p * (\log(n) + 1)} \tag{7}$$

Equation 7 cannot be simplified further, as we did for the ratio of C_{basic}/C_{flat}. The analysis for the basic PCIR is based on the random assignment of peers to groups. This assumption is not valid for the clustering-enhanced approach, where each peer decides on the cluster to join based on its collection. Thus, the characteristic value of β for each cluster collection at the clustering-enhanced PCIR is significantly different than the value of β for a single peer document collection.

6. Experimental evaluation

In addition to the theoretical cost analysis, we also conducted a large-scale experimental evaluation of PCIR. The objective of the experiments was to evaluate the two PCIR variants on real-world datasets with respect to efficiency, and to compare them with the current state-of-the-art approaches for DHT publishing. The chosen experimental configurations cover a wide range of application scenarios, and thoroughly investigate the suitability of the PCIR variants for different system and network configurations.

6.1. Experimental setup and evaluation criteria

The efficiency of the two PCIR algorithms was experimentally evaluated using collections of real documents. In particular, we simulated P2P networks of up to 5000 peers running the two PCIR variants, and measured the total network cost for each algorithm to maintain the DHT inverted index. As a baseline we have used the flat DHT algorithm, according to which every peer publishes its own inverted index in the DHT. For the simulation we considered networks of size N in the range $1000 \leq N \leq 5000$.

We have repeated all experimental setups with and without churn. For the scenarios without churn, we kept the peers and their contents static throughout the experiment. For the scenarios with churn, at each iteration we selected randomly up to 20% of the peers and replaced them with an equal number of new peers, carrying new documents. To make the experimental results independent of the churn factor, we restricted the peers in PCIR in the following ways: (a) peers and super peers did not use delta updating, and, (b) peers were forced to find and rejoin a peer group or a peer cluster at every iteration. Under these

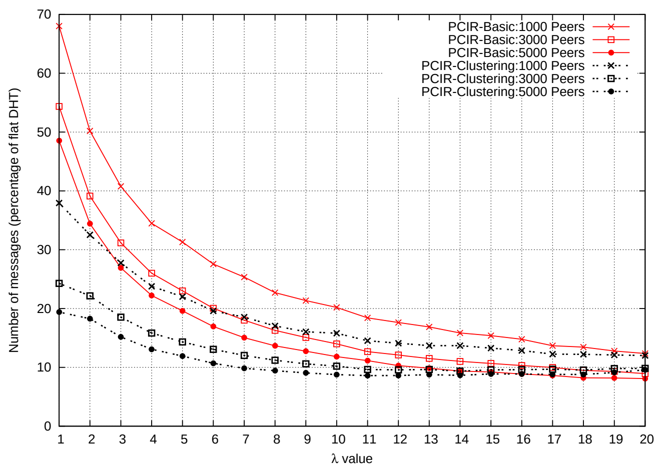


Figure 5: Number of messages for PCIR with upper bound=40000 terms per super peer.

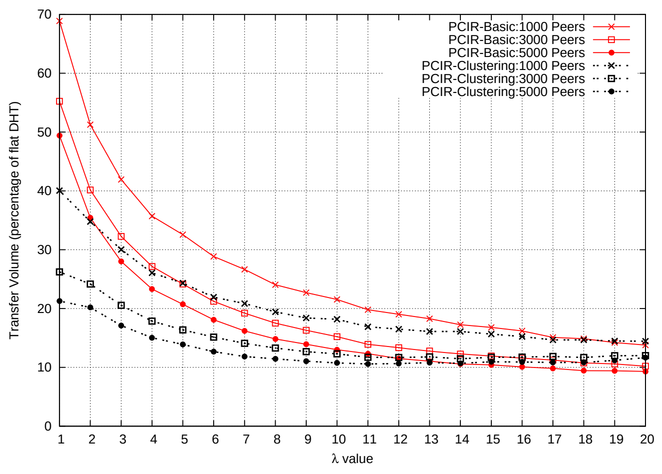


Figure 6: Transfer volume for PCIR with upper bound=40000 terms per super peer.

restrictions, churn had no effect on the results. The results included in this section correspond to the configuration with 20% churn.

The main body of experiments was performed using peer granularity indexing, which is the most widely used. Furthermore, to confirm the general applicability of PCIR for maintaining different types of metadata in the DHT and enabling different information retrieval techniques, we use PCIR to support three additional IR techniques over PCIR: (a) document-granularity indexing [24], also employed in other popular P2P systems, e.g., ALVIS [4] (b) sk-STAT [13], which uses peer-granularity indexing with additional meta-data, (c) mk-STAT [13], a query-driven indexing enhancement of sk-STAT, which includes in the inverted index also some frequently-queried multi-term keys.

For the PCIR approach we varied the following parameters:

- *Top-λ terms, top-μ super peers:* We repeated the experiments for $\lambda = [1, 2, \ldots, 20]$ and $\mu = [1, 2, \ldots, 20]$.
- *Upper bound for super peer workload:* We experimented with upper bounds ranging from 5000 terms to 40000 terms per super peer.

For clustering-enhanced PCIR, we set the Bloom filter length and number of hash functions so that the Bloom filter error probability never exceeded 10%. In particular, we found the optimal Bloom filter length and number of hash functions for each configuration by assuming that the number of objects in the Bloom filters is equal to the upper bound for the super peer workload (5000 to 40000). We also ensured that for the same upper bound, both basic and clustering-enhanced PCIR create the same number of groups/clusters so that super peers in the two approaches represent the same number of peers on average. Since the number of clusters in the clustering-enhanced approach is dynamically determined, at each repetition we first executed the clustering-enhanced setup and then initialized the basic PCIR with the same number of groups.

In the clustering-enhanced PCIR experiments, all peers partitioned their collection to three virtual peers by running K-Means. However, it is not required by the algorithm that all peers are partitioned to an equal number of virtual peers. The number of document categories in real-life peers is expected to vary, and the user, or the partitioning algorithm itself, may decide on a different number of virtual peers.

Construction of peer collections. All experiments were conducted on two datasets, the REUTERS Corpus Volume I (RCV1) [25] and the MEDLINE collection [26]. The results were very similar for the two datasets, so we present only the details for the REUTERS dataset. The complete REUTERS collection includes more than 800,000 categorized newswire articles, pre-processed with stopword filtering and stemming. For the evaluation we used a subset of 160,000 randomly selected articles.

Real-life peer collections, similar to real persons' interests, are often multi-thematic. Some users may be well-focused, having very specific documents of only one topic. Other users may focus on a couple of non-related topics, and yet others may just collect lots of diverse documents. We simulated all such users by using the classification which accompanies the REUTERS document collection. The documents used in the experiments belonged to a total of 148 categories. Peers were creating their collections by: (a) randomly selecting three random categories, and, (b) randomly selecting 20 documents for each of these categories. At the end, each peer had exactly 60 distinct documents. Since the REUTERS classification had categories of different specificities, some peers ended up having many documents of diverse topics, while other peers were focused on three or less very specific topics and had very similar documents overall.

Evaluation criteria. We compare the index maintenance costs of basic and clustering-enhanced PCIR with the costs of the flat DHT approach, for all publishing strategies (peer and document granularity, sk-STAT, and mk-STAT). We measure the number of messages as well as the total data transfer volume caused by each algorithm as follows:

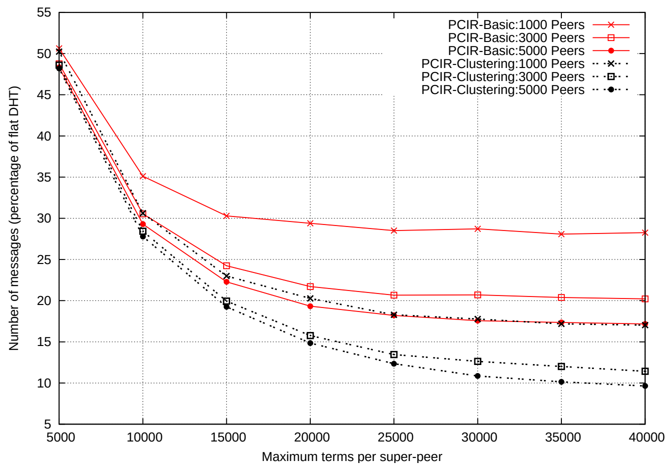


Figure 7: Number of messages for PCIR: $\lambda = 6, \mu = 2$ and varying upper bound.

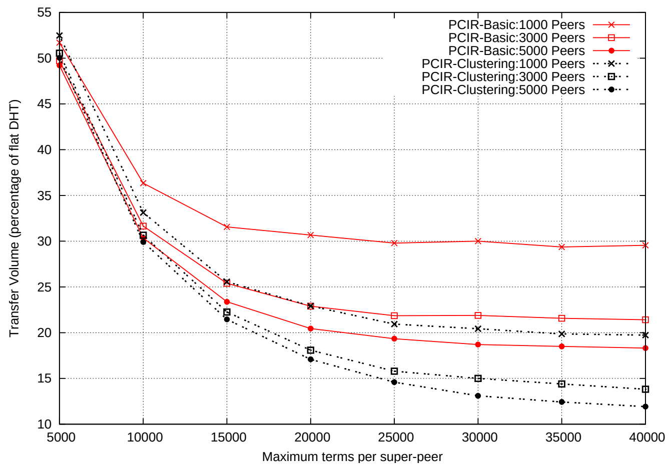


Figure 8: Transfer volume for PCIR: $\lambda = 6, \mu = 2$ and varying upper bound.

- *Number of messages:* We count all messages exchanged in the system, including all DHT and super peer related messages.
- *Transfer volume:* We measure total transfer volume. Apart from the message body (the actual data), we include a network header overhead of 64 bytes for each transaction (theoretical minimum for TCP/IP network transaction). GZIP compression is applied to the message body whenever this reduces the message size.

The described experimental setups were repeated 6 times, and each execution was let to run for 4 iterations (4 DHT periodic submissions). In the following sections we report average results over 6 repeats. Section 6.2 focuses on the experimental results for peer-granularity indexing. Finally, in Section 6.3 we describe the experimental setup and the results corresponding to the three additional publishing strategies.

6.2. Results for peer-granularity indexing

We now report the experimental results for peer-granularity indexing (see Section 3 for a brief description), as used by, e.g., Minerva [5]. Peer granularity indexing is the most widely used, since it provides a favorable compromise between the IR quality and the network cost for indexing [12].

Figures 5 and 6 plot the number of messages and transfer volume required by each approach for different network sizes. X-axis shows the λ value and Y-axis shows the network resources required by each approach, averaged over 6 runs. The cost of flat DHT submission is used as a baseline (always 100% because it is independent of λ). Basic PCIR costs are also not directly relevant to λ; they are related indirectly to λ via the number of groups (the number of groups at basic PCIR was equal to the number of clusters at clustering-enhanced PCIR, which was dependent on λ).

The results are for an upper bound of 40000 terms per super peer. For clarity we include only results for 1000, 3000, and 5000 peers. In these experiments, the 40000 terms upper bound is never reached, so the results for the experiment without an upper bound are the same. Parameter λ takes values from 1 to 20, and μ is set to one. Because of the high upper bound of terms per cluster in this experiment, there is never more than one candidate cluster for a joining peer. Thus, increasing μ to more than one has no effect in the results for this case.

Network savings grow with network size (Fig. 5 and 6). For $\lambda \leq 10$, clustering-enhanced PCIR is significantly better than basic PCIR. Clustering-enhanced PCIR reaches its optimal performance for $\lambda \approx 8$. Very low λ values ($\lambda \leq 3$) result in the creation of some small peer clusters and limit the effectiveness of the peer clustering algorithm. 90% of the optimal performance of clustering-enhanced PCIR is already obtained with $\lambda = 5$. For $6 \leq \lambda \leq 10$, the performance of clustering-enhanced PCIR stays almost unchanged. For $\lambda > 10$, the two PCIR variants have comparable performance. For very high λ, the overall number of messages and transfer volume at clustering-enhanced PCIR increases because of the increase in the cost for a peer to find and join a cluster.

Finding the optimal λ and μ values. The λ and μ values for which clustering-enhanced PCIR minimizes the network costs depend on the network size, upper bound, and on the collection (see Section 5). An online optimization similar to [27] can be used for optimizing the values and minimizing the cost. However, our experiments show that in practice this is not necessary, since varying λ between 5 and 10 has only a small effects on performance, and increasing μ beyond 2 is not beneficial. For all experiments, a setup with $\lambda = 6$ and $\mu = 2$ achieved at least 90% of the savings obtained by the optimal configuration. Especially for the larger networks (i.e., 3000 peers and more), the configuration with $\lambda = 6$ and $\mu = 2$ achieved more than 95% of the optimal performance. The difference between the optimal-minimal cost and the cost of PCIR with $\lambda = 6$ and $\mu = 2$ was decreasing with the increase of network size.

From these results we conclude that online optimization for the values of λ and μ is not necessary, and constant values of $\lambda = 6$ and $\mu = 2$ give near optimal results. The rest of the results presented in this section are for $\lambda = 6$ and $\mu = 2$.

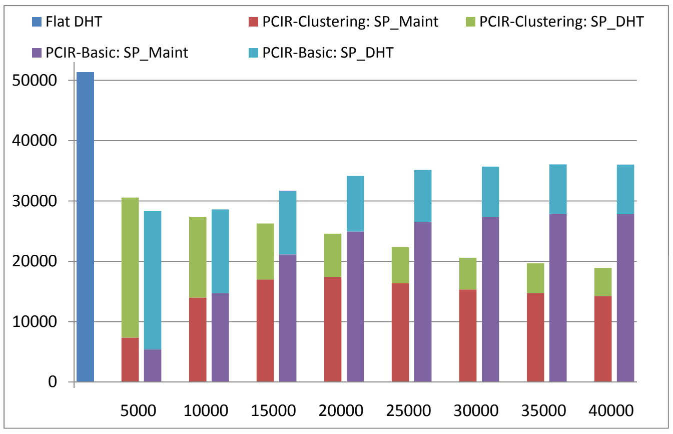


Figure 9: Number of messages for PCIR super peers: $\lambda = 6, \mu = 2$ and varying upper bound.

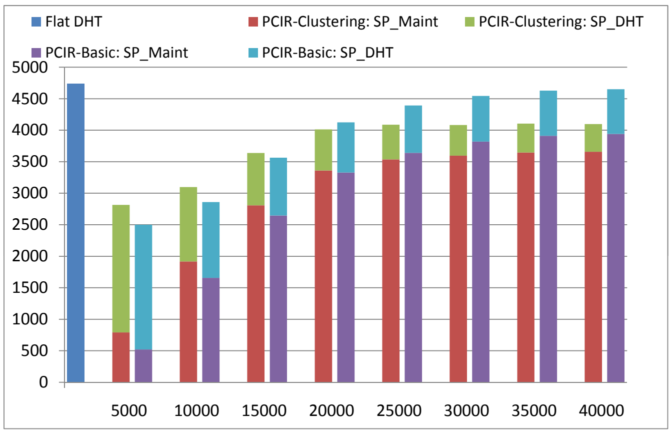


Figure 10: Transfer volume for PCIR super peers in Kbytes: $\lambda = 6, \mu = 2$ and varying upper bound.

Varying the maximum terms per super peer. In practice, each super peer sets its upper bound itself, based on the network resources it can offer. A large upper bound enables better clustering and higher term overlap. In our experiments, we assume that all the super peers have the same network resources and set their upper bound to the same value.

Figures 7 and 8 present the total maintenance cost (for both peers and super peers) for different upper bound values, for $\lambda = 6$ and $\mu = 2$. For an upper bound of 5000 terms, clustering-enhanced PCIR has the same performance as the basic PCIR. This is expected since such a low upper bound is reached already by the document collections of six peers on average. Therefore, peer clustering cannot effectively increase the term overlap in this case. With larger upper bounds, term overlap in the clusters increases, and less clusters are required overall. Increasing the upper bound beyond 20000 terms further reduces the total cost but at a slower rate.

It is also interesting to see the ratio between physical peers and super peers for the case of clustering-enhanced PCIR, and to investigate how this corresponds to the upper bound per super peer. Table 2 includes example values for the network of 5000 peers, with different upper bounds. As expected, increasing the upper bound clearly results to less super peers. However, at some point, i.e., after 20000 terms, the number of super peers does not change significantly. This denotes that it is infrequent that the super peers acquire a larger dictionary, and it happens for two reasons. First, the dictionary size per super peer follows the Heap's law (see Section 5), and therefore it grows slowly with the number of peers and documents in the cluster. Second, the virtual peers and documents are assigned to each super peer such that the term overlap is increased, i.e., using clustering. Therefore, each peer contributes only a small number of new terms in its super peer. As such, the upper bound per super peer does not need to be large for PCIR to be efficient.

Super peers workload. To ensure that super peers do not constitute a bottleneck for PCIR we also count the network workload of super peers in PCIR. For clustering-enhanced PCIR, we measure the total workload of the physical peer which hosts the

Upper bound	#Super peers	Ratio
5000	844	0.168
10000	278	0.055
15000	156	0.031
20000	118	0.023
25000	104	0.020
30000	98	0.019
35000	96	0.019
40000	93	0.018

Table 2: Number of super peers, and ratio of super peers:total physical peers for different upper bounds, for a network of 5000 peers

super peer and 2 regular virtual peers. For the basic PCIR approach, since the physical peers host exactly one virtual peer (super peer or regular peer), the workload of the physical peer includes only the workload of the super peer.

For both PCIR variants we distinguish between two types of workload:

SP_{Maint}: Workload that is generated because of the role of the peer as super peer, i.e., for maintaining the peer groups/clusters, and for publishing the peer scores for all group/cluster terms.

SP_{DHT}: Workload generated because the super peer participates in the DHT, i.e., for routing DHT lookups which are issued from other peers.

Figures 9 and 10 plot the network workload of super peers at the largest setup with 5000 peers. The results are for $\lambda = 6$ and $\mu = 2$ (the configuration which gives at least 90% of the optimal performance for all setups). X-axis shows the upper bound of terms per super peer and Y-axis corresponds to the average network workload per super peer. For the clustering-enhanced PCIR, the super peer workload includes the total workload of the physical peer, i.e., the workload generated by the super peer and by the two regular virtual peers. For the basic PCIR approach, since the physical peer hosts only one virtual peer (super peer or regular peer), the workload of the physical peer includes only the workload of the super peer. For reference, the figure also includes the load per physical peer in the flat DHT approach which maintains the same DHT index.

In all experiments, average network workload of super peers at both PCIR variants is less than the respective workload of regular peers at the flat DHT indexing approach. Regarding the number of messages, super peers at both PCIR variants have significantly less workload compared to regular peers at flat DHT indexing. Regarding transfer volume, super peers at basic PCIR have comparable cost with the physical peers at flat DHT indexing, while super peers at clustering-enhanced PCIR still have substantially less transfer volume compared to the physical peers at flat DHT indexing.

Figure 9 shows that for the case of clustering-enhanced PCIR, the number of messages per super peer decreases with the upper-bound per super peer, even though super peers are burdened with indexing more terms. This happens because with higher upper bounds, peer clustering achieves higher term overlap in the super peers. With higher term overlap, fewer DHT lookups are created for indexing all peers, and fewer DHT lookup messages need to be routed over the DHT. Therefore, the number of messages is substantially reduced for all participating peers (both regular peers and super peers). The network savings per super peer attributed to this reduction (reduction in cost SP_{DHT}) are more than the additional cost imposed at the super peer because of the higher upper bound (increase in cost SP_{Maint}). As a result, the total number of messages per super peer is reduced with an increase of the upper bound.

PCIR super peers also incur less transfer volume compared to the flat DHT peers, even for the maximum upper bound of 40000 terms per super peer (Figure 10). For low upper bound values – less than 20000 terms – super peers in basic and clustering-enhanced PCIR have comparable transfer volume. However, for higher upper bounds – more than 25000 terms – peer clustering becomes more effective, and super peers in clustering-enhanced PCIR end up with significantly less transfer volume compared to super peers in basic PCIR. We also see that in contrast to number of messages, transfer volume per super peer increases when the upper bound is increased. This is expected because for a higher upper bound, more virtual peers are allowed to join each super peer. Although each virtual peer contributes only a small number of messages for its super peer, it needs to send its complete inverted index to its super peer, and this increases the transfer volume of the super peer. However, as explained in Section 3, each super peer can choose the upper bound for its workload independently so that its total workload does not exceed its capabilities.

Finally, for upper bounds higher than 30000 terms, the cost related to the super peer role (cost SP_{Maint}) does not change significantly, meaning that cluster sizes of most super peers never reach the higher upper bounds. This observation is also confirmed from the results in Table 2, which show that by increasing the upper bound to more than 30000 terms, only 5 clusters are affected. According to clustering-enhanced PCIR algorithm, when peers are not similar enough to be clustered together, new clusters are created, so that centroids of existing clusters do not shift. Only super peers that have chosen a high upper bound and also have a centroid which is very similar to many peer centroids have an increase in their workload.

To keep the results independent of peer collection updates, peers did not use delta updating in the aforementioned experiments. In real-world deployments, virtual peers would not be required to send their whole inverted index at their super peers at each step. They could instead update the inverted indices at their super peers sending only their changes. This would significantly decrease the overall network load, and especially the transfer volume of the super peers. Note that delta updating is not beneficial for flat DHT publishing, where the messages are generally very small.

Cost estimation accuracy. The experimental results also support the cost equations presented in Section 5. Accuracy of the estimations was confirmed as follows. We first incrementally sampled a small number of peers (0.3% of the total peers), and counted the distinct and total terms for each sample. By applying the Gauss-Newton method for nonlinear fitting we estimated the value of β for the text collection (0.59 for Medline and 0.55 for the Reuters collection). Then we ran the flat DHT and the basic PCIR approach for all network sizes and measured the number of messages required by the two approaches. We also computed the expected cost ratio C_{basic}/C_{flat}, based on Theorem 5.1. The cost ratio computed experimentally was very close to the expected cost ratio. In particular, the maximum difference between the expected and experimental value was less than 7%, and the average difference was around 4%. Especially regarding networks larger than 3000 peers, average difference was reduced to less than 3% because of the larger sample size for estimating β. Nevertheless, even for the largest network of 5000 peers, sampling was still inexpensive as it involved only 15 peers.

Summary: The experimental results demonstrate that both PCIR approaches are significantly more efficient than flat DHT indexing. For large network sizes, cost for generating exactly the same inverted index with cluster-based PCIR is an order of magnitude less than the corresponding cost with flat DHT indexing. Network workload per super peer is also less than network workload of regular peers in the flat DHT indexing approach.

6.3. Results for additional publishing strategies

To confirm the general applicability of PCIR for optimizing different indexing strategies, we also implemented three additional index publishing strategies over PCIR. The first one constructs document-granularity inverted indices, for increased retrieval quality [24]. Variants of document granularity indexing are used in well-known P2P systems, e.g., ALVIS [4], therefore it is important to show that PCIR can also support these systems. The second and third approach are sk-STAT and mk-STAT [13]. Both approaches employ peer-granularity indices, similar to Minerva [5]. However, the inverted index in sk-STAT includes additional term statistics per peer using hash sketch synopses, which are used to improve IR precision and efficiency. In addition to this, mk-STAT uses query logs to identify and index in the DHT interesting term combinations, which

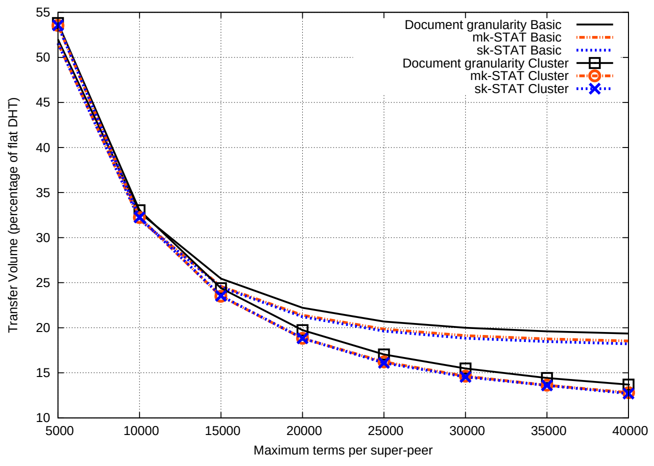


Figure 11: Transfer volume for different publishing strategies with PCIR

are used for answering multi-term queries efficiently. The three publishing strategies are described in more details in Section 7.

For this experiment, we have implemented the three publishing strategies as described in the original papers, and simulated them in our experimental setup. Each approach was simulated in two variants: (a) where each peer publishes its own information (i.e., flat DHT indexing), as proposed in the corresponding original works, and, (b) where the PCIR algorithm is applied to reduce index maintenance cost. mk-STAT requires a query log to identify frequent term combinations. We have used the AOL query log to simulate queries, and configured the algorithm to consider the 25% most frequent queries as candidates for multi-term indexing. The conditional probability threshold for indexing a multi-term was set to 0.1, as proposed in [13]. PCIR was configured with $\lambda = 6$ and $\mu = 2$.

Figure 11 presents the transfer volume for the three publishing strategies, for different upper workload bounds for the super peers. The presented results are for a network of 5000 peers. For each publishing strategy, the transfer volume of PCIR is presented as a percentage of the required transfer volume for flat DHT indexing for the particular publishing strategy, as incurred by [4, 13]. We see that PCIR offers substantial performance improvements with respect to the three alternative publishing strategies. The gain is comparable to the peer-granularity strategy, discussed in Section 6.2. The clustering-enhanced PCIR variant is again significantly better than its basic counterpart. Similar to the results for the peer-granularity strategy, the network gains increase with the super peers upper bound, up to an order of magnitude lower cost compared to the baselines for upper bounds higher than 30000 terms.

The number of messages required by the three alternative publishing strategies is equal to the peer-granularity strategy, as reported in Section 6.2. As explained in [13], this also holds for mk-STAT, even though it maintains additional multi-term keys in the DHT, because the additional information is piggy-packed on single-term maintenance messages.

Summary: The experimental results with additional publishing strategies confirm that PCIR is equally effective when used for document-granularity publishing, query-driven publishing, and sk-STAT. Independent of the publishing strategy, clustering-enhanced PCIR imposes an order of magnitude less transfer volume in the participating peers compared to the standard approach where each peer publishes the same information for itself.

7. Related Work

7.1. P2P Information Retrieval

P2P information retrieval has been studied in a number of previous publications. The first proposals focused on distributed IR in unstructured networks, using approximated system-wide information. PlanetP [28] is one of these systems. It uses gossiping to distribute peer content summaries, represented by Bloom filters, to all participating peers. From these summaries each peer computes peer frequencies, which are used to rank peers for a given query without central coordination. Due to the usage of gossiping for distribution of collection-wide information, PlanetP and similar unstructured P2P approaches exhibit only limited scalability.

To overcome these limitations, super peer topologies were proposed, where dedicated nodes take the responsibility for index maintenance and access[29, 30]. While these systems scale better, they require peers with very good network connections and high availability to cope with the workload imposed by these tasks.

P2P IR systems based on DHTs avoid this load balancing issue and promise high scalability due to the fact that DHT costs only grow logarithmically with network size [31, 4, 5]. As explained in Section 2, these systems enable information retrieval by constructing a distributed inverted term index. As shown, the main weakness of DHT-based P2P IR systems is the high effort for maintenance of the inverted index. Previous systems vary granularity and completeness of the inverted index, resulting in different quality/cost tradeoffs. In the ALVIS system [31, 4], peers index the terms for each of their documents independently (document-granularity index). In this way ALVIS achieves high performance query execution and high information retrieval quality, but at a high cost for the index maintenance. In contrast, peer-granularity systems trade index accuracy with maintenance cost. For example, the peers in Minerva [5, 6] aggregate their documents' scores per term to produce a peer score for each term. To address the inability of collecting document-granularity statistics that can improve the IR quality, sk-STAT [13] which builds on top of Minerva, enhances the peer scores with all document identifiers having each term (e.g., their file names), represented as hash sketches. Although maintenance cost for peer-granularity systems is lower compared to document-granularity systems, the approach still does not scale for full-text indexing because the total number of distinct terms per peer, and the number of DHT lookups, are not reduced. In fact, Li et al. [7] have shown that a P2P solution cannot scale to a large network size if full-text indexing is used, mainly because index maintenance becomes too expensive. Our experiments, presented in Section 6, further suggested that the main fraction of network cost is generated by the huge number of DHT lookups, which is independent of the granularity of the inverted index, and that by reducing the DHT lookups one can efficiently reduce the overall index maintenance cost.

The Adlib approach [32] follows a different direction for reducing network cost. It establishes a two-tier structure, where a first tier divides the documents into independent, equal-sized partitions, called domains. Within each domain, nodes build a distributed index for the documents, which is then offered in the second tier for querying. By tuning size and number of domains, index maintenance can be traded against query efficiency. This approach reduces the number of DHT lookups, but only with trade-offs for query execution, either in the quality of the results or in the query cost. If full-text querying over the whole P2P network is required, Adlib is less efficient than traditional flat DHT indexing.

Another way to reduce network costs is to index only a subset of terms occurring in a peer's collection. Peers in PNear [9] randomly choose a subset of the terms to be indexed. Crespo and Garcia-Molina [33] organize the peers into semantic overlay networks by asking each user to manually select the terms for her files. The proposal by Loo et al. [8] builds a hybrid Gnutella/DHT infrastructure to limit the network cost. Only rare items are indexed in the DHT, while search for frequent items is done via message flooding in the Gnutella topology. In general, the systems in this family reduce DHT maintenance cost significantly, as peers do not publish all their terms. However, because of the incomplete index, retrieval quality is reduced.

Nguyen et al. [34] follow a different approach for reducing the inverted index maintenance cost, called adaptive distributed indexing. Instead of indexing all terms at a preselected granularity – either peer granularity or document granularity – each peer forms small groups of its documents and indexes the terms of each group as a large virtual document. The publishing granularity, i.e., the size of the document groups, is selected such that overall cost for index maintenance and query execution is reduced. In contrast to PCIR, adaptive distributed indexing does not focus on reducing the number of DHT lookups which constitute the major indexing cost. In fact, number of DHT lookups is orthogonal to the indexing granularity level. Therefore, adaptive distributed indexing can also benefit from PCIR, for reducing the DHT lookups, and thereby drastically reducing the total indexing cost.

The concept of query-driven indexing has also been recently exploited. For example, in a recent version of ALVIS [14], peers identify frequent multi-term queries, and cache their results in the DHT. This offers faster query execution with lower network overhead, albeit at the expense of a larger inverted index over the DHT. To reduce this additional cost, mk-STAT [13], another query-driven indexing approach, imposes additional constraints on what constitutes an interesting multi-term query: terms that are highly statistically correlated in the documents are not considered interesting, since querying for one of the terms alone already returns most of the results. As we have already shown in Section 6.3, PCIR also enables substantial network savings for query driven indexing.

Hierarchical DHTs have also been introduced, as a mean to foster efficient bandwidth utilization and a better adaptation to the underlying physical network [35, 36]. The latter point is promising, especially for information retrieval tasks in peer-to-peer networks. However, a theoretical analysis shows that current proposals for hierarchical DHTs are still less effective in number of messages compared to flat DHTs [36].

The issue of too many DHT accesses can also be partially alleviated by buffering and aggregating small DHT messages with the same destination [37]. This optimization is orthogonal to the term indexing strategy and can be used to further optimize any approach, including the one proposed in this work. In fact, PCIR also generates small messages – DHT lookups – therefore it can naturally benefit from message aggregation, as long as this aggregation does not cause performance issues for the system. Examining the benefits of integrating PCIR with message aggregation at the network layer is part of our current work.

In this work we have applied PCIR on two different peer-granularity publishing strategies [13, 5], a query-driven strategy [13], as well as a document-granularity strategy [24]. However, the application of our approach to other DHT-based IR systems is straightforward. For example, a recent version of ALVIS [38] increases query execution efficiency by changing the way documents are indexed: Instead of publishing only the term scores for each document, each peer identifies the highly discriminative keys from each document – each key can consist of more than one term – and uses these keys to index the document. For indexing all highly discriminative keys, the number of required DHT lookups is increased. An indexing scheme such as the one proposed by PCIR can effectively reduce the indexing cost for this approach too, without interfering with the information retrieval quality or with the query execution efficiency. Other DHT-based systems that perform full-text indexing, e.g., [34], can also employ PCIR for reducing the network cost, without affecting their query execution part.

7.2. Distributed peer clustering

In this work we developed a P2P clustering algorithm which enables peers that share similar collections to group together in a P2P network. Although the focus of this work is not on proposing a high-quality P2P clustering algorithm, an inexpensive and effective peer clustering algorithm is important for the performance of PCIR. Some previous approaches exist for P2P clustering, but none of them is suitable for our setup. Hammouda and Kamel [39] use a hierarchical topology for the coordination of K-Means computation. Local K-Means is performed at each hierarchy level, and the clustering results are merged hierarchically to produce the final centroids. However, the reported experiments already show that the algorithm suffers in large networks; already for 65 peers the F-Measure quality drops to less than 20% of the quality achieved by the centralized K-Means, making this algorithm unsuitable for large P2P systems.

The P2P K-Means algorithm, proposed by Datta et al. [40], is based on gossiping. Each peer performs local K-Means clustering and then uses gossiping to send its centroids at all its neighbors. It averages the centroids received from its neighbors with its own centroids to produce its new centroids, and repeats the process until the centroids do not change significantly between two iterations. The algorithm is accurate and inexpensive for low dimensional data. However, as shown in [41], this algorithm fails on clustering high-dimensional data such as text.

8. Conclusions and Outlook

In this paper we presented a hybrid DHT/super peer system that significantly reduces network cost for maintaining a complete inverted index. Because individually publishing all terms of each peer's vocabulary to the DHT is too expensive, we group peers around super peers that are responsible for integrating all information about their group's collections, and for publishing it to the DHT. The network gain is achieved by exploiting the term overlap at the super peers to reduce expensive DHT lookups. Experiments with real-world datasets show a gain of an order of magnitude over the conventional flat DHT approach. A theoretical evaluation explains the benefits of our approach and supports our experimental results.

The two algorithms presented in this paper differ in the way peers build groups around super peers: the basic algorithm groups peers randomly, whereas the clustering-enhanced algorithm first partitions peers into thematically focused virtual peers that can then be clustered more effectively, to foster group homogeneity. Whether creating random groups of peers or using clustering algorithms for grouping, a significant term overlap within each group is observed. Significant network savings are observed in both algorithms, regarding both number of messages and transfer volume. Compared to the basic algorithm, clustering-enhanced PCIR increases the term overlap in each group significantly, and thus requires even less DHT lookups than the basic approach. Both approaches reduce the DHT maintenance network cost without having negative effects in the querying phase, regarding both precision/recall as well as performance of the original DHT model.

Super peers in our PCIR approaches do not get overloaded; the maximum workload of the super peers can be set by the super peers themselves. Experiments without upper bounds show that super peers still have less workload compared to the workload of regular peers in conventional DHT publishing.

Our future work will address mainly two topics, implementing more accurate peer clustering, and designing advanced information retrieval techniques.

Improved Peer Clustering. We expect that an improved clustering of peers to clusters will further increase term overlap and reduce the distinct terms per cluster. Hence, we will evaluate more complex distributed clustering algorithms and study their execution cost in our DHT scenario.

Information retrieval techniques. The proposed PCIR cluster-based model enables novel techniques for P2P information retrieval. Using intra-cluster communication, peers in a cluster can efficiently execute information retrieval techniques that go beyond traditional $TF \times IDF$, which are otherwise too complex to run in a distributed fashion. Consider for instance a two-step query execution technique: (a) first, find a proper cluster, and (b) second, find the right peers to query within the cluster. For the second step, since the number of peers per cluster is limited (as opposed to the total number of peers in the network), even complex information retrieval infrastructures can be employed for query evaluation within each cluster. At the same time, we can limit the inverted index over the global DHT to include only aggregated cluster data, and therefore reduce further the DHT indexing cost.

References

[1] M. Mani, A.-M. Nguyen, N. Crespi, Scope: A prototype for spontaneous p2p social networking, in: Proc. International Workshop on Communication, Collaboration and Social Networking in Pervasive Computing Environments, 2010.

[2] M. Mani, A.-M. Nguyen, N. Crespi, What's up 2.0: P2p spontaneous social networking, in: IEEE INFOCOM, Poster session, 2009.

[3] G. Oster, P. Molli, S. Dumitriu, R. Mondéjar, Uniwiki: A collaborative p2p system for distributed wiki applications, in: Fifth International Workshop on Collaborative Peer-to-Peer Systems, 2009.

[4] T. Luu, G. Skobeltsyn, F. Klemm, M. Puh, I. P. Žarko, M. Rajman, K. Aberer, AlvisP2P: scalable peer-to-peer text retrieval in a structured P2P network, Proc. VLDB Endow. 1 (2) (2008) 1424–1427.

[5] M. Bender, S. Michel, P. Triantafillou, G. Weikum, C. Zimmer, Improving collection selection with overlap awareness in P2P search engines, in: Proc. 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2005.

[6] M. Bender, S. Michel, P. Triantafillou, G. Weikum, C. Zimmer, Minerva: Collaborative p2p search, in: VLDB, 2005, pp. 1263–1266.

[7] J. Li, B. T. Loo, J. M. Hellerstein, M. F. Kaashoek, D. R. Karger, R. Morris, On the feasibility of peer-to-peer web indexing and search, in: Proc. Second International Workshop on Peer-to-Peer Systems (IPTPS), La Jolla, CA, USA, 2003.

[8] B. T. Loo, R. Huebsch, I. Stoica, J. M. Hellerstein, The case for a hybrid P2P search infrastructure, in: Proc. Third International Workshop on Peer-to-Peer Systems (IPTPS), La Jolla, CA, USA, 2004.

[9] R. Siebes, pNear: combining content clustering and distributed hash tables, in: Proc. Second International Workshop on Peer-to-Peer Knowledge Management (P2PKM), 2005.

[10] O. Papapetrou, W. Siberski, W.-T. Balke, W. Nejdl, Dhts over peer clusters for distributed information retrieval, in: Proc. 21st International Conference on Advanced Information Networking and Applications (AINA), 2007, pp. 84–93.

[11] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, SIGCOMM Comput. Commun. Rev. 31 (4) (2001) 149–160.

[12] J. P. Callan, Z. Lu, W. B. Croft, Searching Distributed Collections with Inference Networks, in: Proc. 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM Press, Seattle, Washington, 1995.

[13] S. Michel, M. Bender, N. Ntarmos, P. Triantafillou, G. Weikum, C. Zimmer, Discovering and exploiting keyword and attribute-value co-occurrences to improve p2p routing indices, in: CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management, ACM, New York, NY, USA, 2006, pp. 172–181. doi:http://doi.acm.org/10.1145/1183614.1183643.

[14] G. Skobeltsyn, T. Luu, I. P. Zarko, M. Rajman, K. Aberer, Query-driven indexing for scalable peer-to-peer text retrieval, Future Generation Comp. Syst. 25 (1) (2009) 89–99.

[15] L. Massoulie, E. L. Merrer, A.-M. Kermarrec, A. Ganesh, Peer counting and sampling in overlay networks: random walk methods, in: Proc. 25th annual ACM symposium on Principles of distributed computing, 2006.

[16] B. H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications ACM 13 (7) (1970) 422–426.

[17] L. Fan, P. Cao, J. Almeida, A. Z. Broder, Summary cache: a scalable wide-area Web cache sharing protocol, IEEE/ACM Transactions on Networking 8 (3) (2000) 281–293.

[18] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmr, T. Risch, EDUTELLA: A P2P Networking Infrastructure based on RDF, in: Proc. 11th International World Wide Web Conference (WWW2002), Hawaii, USA, 2002.

[19] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, R. Schmidt, P-Grid: A self-organizing structured P2P system, SIGMOD Record 32 (3) (2003) 29–33.

[20] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, R. Morris, Designing a DHT for low latency and high throughput, in: Proc. 1st Symposium on Networked Systems Design and Implementation, 2004.

[21] L. Lee, Measures of distributional similarity, in: Proc. 27th Annual Meeting of the Association for Computational Linguistics, 1999.

[22] C. Blake, A comparison of document, sentence, and term event spaces, in: Proc. 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL, Association for Computational Linguistics, Morristown, NJ, USA, 2006.

[23] H. S. Heaps, Information Retrieval – Computational and Theoretical Aspects, Academic Press, 1978.

[24] P. Reynolds, A. Vahdat, Efficient peer-to-peer keyword searching, in: Middleware, Vol. 2672 of Lecture Notes in Computer Science, Springer, 2003, pp. 21–40.

[25] D. D. Lewis, Y. Yang, T. G. Rose, F. Li, RCV1: A new benchmark collection for text categorization research, J. Mach. Learn. Res. 5 (2004) 361–397.

[26] Medline database, US National Library of Medicine, http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?DB=pubmed (2006).

[27] L. Xiao, Y. Liu, L. M. Ni, Improving unstructured peer-to-peer systems by adaptive connection establishment, IEEE Trans. Comput. 54 (9) (2005) 1091–1103.

[28] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, T. D. Nguyen, PlanetP: Using gossiping to build content addressable peer-to-peer information sharing communities, in: Proc. Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), 2003.

[29] W.-T. Balke, W. Nejdl, W. Siberski, U. Thaden, Progressive distributed top-k retrieval in peer-to-peer networks, in: Proc. 21st International Conference on Data Engineering (ICDE), Tokio, Japan, 2005.

[30] B. Cooper, Guiding queries to information sources with InfoBeacons, in: Proc. ACM/IFIP/USENIX 5th International Middleware Conference, ACM, 2004.

[31] K. Aberer, F. Klemm, M. Rajman, J. Wu, An architecture for peer-to-peer information retrieval, in: Proc. Workshop on Peer-to-Peer Information Retrieval, 2004.

[32] P. Ganesan, Q. Sun, H. Garcia-Molina, Adlib: A self-tuning index for dynamic peer-to-peer systems, in: Proc. 21st International Conference on Data Engineering, ICDE 2005, IEEE, 2005.

[33] A. Crespo, H. Garcia-Molina, Semantic overlay networks for P2P systems, in: AP2PC, 2004, pp. 1–13.

[34] L. T. Nguyen, W. G. Yee, O. Frieder, Adaptive distributed indexing for structured peer-to-peer networks, in: J. G. Shanahan, S. Amer-Yahia, I. Manolescu, Y. Zhang, D. A. Evans, A. Kolcz, K.-S. Choi, A. Chowdhury (Eds.), CIKM, ACM, 2008, pp. 1241–1250.

[35] P. Ganesan, P. K. Gummadi, H. Garcia-Molina, Canon in G major: Designing DHTs with hierarchical structure, in: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS), Tokyo, Japan, 2004, pp. 263–272.

[36] S. Zoels, Z. Despotovic, W. Kellerer, Cost-based analysis of hierarchical DHT design, in: Proceedings of the 6th International Conference on Peer-to-Peer Computing, Cambridge, UK, 2006.

[37] F. Klemm, K. Aberer, Aggregation of a term vocabulary for P2P-IR: A DHT stress test, in: DBISP2P, 2005, pp. 187–194.

[38] I. Podnar, M. Rajman, T. Luu, F. Klemm, K. Aberer, Scalable peer-to-peer web retrieval with highly discriminative keys, in: ICDE, 2007, pp. 1096–1105.

[39] K. Hammouda, M. Kamel, HP2PC: Scalable hierarchically-distributed peer-to-peer clustering, in: Proc. 2007 SIAM International Conference on Data Mining (SDM07), Minneapolis, MN, USA, 2007.

[40] S. Datta, C. Giannella, H. Kargupta, K-Means clustering over a large, dynamic network, in: Proc. SDM, 2006.

[41] O. Papapetrou, W. Siberski, F. Leitritz, W. Nejdl, Exploiting distribution skew for scalable p2p text clustering, in: DBISP2P, 2008, pp. 1–12.

[42] L. Michael, W. Nejdl, O. Papapetrou, W. Siberski, Improving distributed join efficiency with extended bloom filter operations, in: 21st International Conference on Advanced Information Networking and Applications (AINA-07), IEEE Computer Society, 2007.

A. Proofs

Proof of Theorem 4.1 We now show how the cardinality of the intersection of two sets A and B can be derived from their Bloom filters. Briefly, the proof proceeds as follows. We will first estimate the number of bits that are set to true in both Bloom filters BF_A and BF_B, but from a different element in each Bloom filter. These bits are set to true due to hash collisions, and by estimating the number of these collisions (Eqn. 8), we can estimate the number of true bits in the Bloom filter of the intersection of the two sets $A \cap B$. From there, we can use a theorem from [42] to estimate the number of elements in the intersection.

To simplify exposition, we represent Bloom filters as sets of numbers. The set representation of a Bloom filter contains value i if and only if the ith bit of the corresponding bit array is true, i.e., $SET_{BF} = \{i : BF[i] = true\}$.

With $BF_\cap$ we denote the Bloom filter of the intersection of the two sets $A \cap B$. $BF_\wedge$ denotes the Bloom filter produced by a bitwise-AND merging of BF_A and BF_B. With R (for random) we denote the number of bits set in BF_A and BF_B, therefore also in $BF_\wedge$, but not set in $BF_\cap$.

The expected value for R, denoted as $\hat{R}$, is found as follows. The elements in $SET_{BF_A} \backslash SET_{BF_\cap}$ are independent from the elements in $SET_{BF_B} \backslash SET_{BF_\cap}$. Thus the probability of an element to occur in both $SET_{BF_A} \backslash SET_{BF_\cap}$ and $SET_{BF_B} \backslash SET_{BF_\cap}$ is $\frac{|SET_{BF_A}|-|SET_{BF_\cap}|}{m-|SET_{BF_\cap}|} * \frac{|SET_{BF_B}|-|SET_{BF_\cap}|}{m-|SET_{BF_\cap}|}$, where $|SET_x|$ denotes the cardinality of SET_x.

When an element occurs in both $SET_{BF_A} \backslash SET_{BF_\cap}$ and $SET_{BF_B} \backslash SET_{BF_\cap}$ it also occurs in $SET_{BF_\wedge}$. The expected value of R is:

$$\begin{aligned} \hat{R} = \quad & (m - |SET_{BF_\cap}|) * \frac{|SET_{BF_A}|-|SET_{BF_\cap}|}{m-|SET_{BF_\cap}|} \\ & * \frac{|SET_{BF_B}|-|SET_{BF_\cap}|}{m-|SET_{BF_\cap}|} \end{aligned} \tag{8}$$

Moreover, by definition:

$$|SET_{BF_\wedge}| = |SET_{BF_\cap}| + R \tag{9}$$

By replacing R in equation 9 with the expected value we get an estimation for $|SET_{BF_\cap}|$:

$$\begin{aligned} E(|SET_{BF_\cap}|) \quad &= |SET_{BF_\wedge}| - (m - |SET_{BF_\cap}|) * \\ & \frac{|SET_{BF_A}|-|SET_{BF_\cap}|}{m-|SET_{BF_\cap}|} * \frac{|SET_{BF_B}|-|SET_{BF_\cap}|}{m-|SET_{BF_\cap}|} \end{aligned} \tag{10}$$

Note that $BF_\cap$ is a normal Bloom filter of the set $A \cap B$. Thus we can use Lemma 4.1 from [42] to estimate the number of objects hashed into it: $|SET_{BF_\cap}| = m * \left(1 - (1 - 1/m)^{kn}\right)$, where $n = E(|A \cap B|)$. Combining that with equation 10, we get an estimation for $E(|A \cap B|)$:

$$
\begin{aligned}
&E(|A \cap B|) = \\
&\quad \frac{\ln(m^2 - m|SET_{BF_A}| - m|SET_{BF_B}| + |SET_{BF_A}| * |SET_{BF_B}|)}{k \ln(1 - 1/m)} - \\
&\quad \frac{\ln(m^2 - m * |SET_{BF_A}| - m * |SET_{BF_B}| + m * |SET_{BF_\wedge}|)}{k \ln(1 - 1/m)}
\end{aligned} \tag{11}
$$

Equation 2 is derived by replacing the notation of $|SET_{BF_x}|$ with the original notation of $tb(BF_x)$. $\blacksquare$

Proof of Theorem 4.2. We use Theorem 4.1 to estimate the expected cardinality of $S_x := P_i \cap C_x$ and $S_y := P_i \cap C_y$. The expected cardinality for the two sets is denoted with $f(P_i, C_x)$ and $f(P_i, C_y)$, and the true (unknown) cardinality is denoted with $|S_x|$ and $|S_y|$.

Without loss of generality assume that $f(P_i, C_x) > f(P_i, C_y)$. In such a case the objective function selects C_x as the optimal one for peer P_i. The objective function selection is wrong when $|S_y| > |S_x|$. We find the probability of $|S_y| > |S_x|$ using Chernoff bounds.

$$
\begin{aligned}
&Pr\Big[|S_y| > |S_x|\Big] < \\
&\quad Pr\Big[|S_y| > (1 + \delta_y) * f(P_i, C_y)] * Pr[|S_x| < (1 - \delta_x) * f(P_i, C_x)\Big] \\
&\quad = exp(-f(P_i, C_y) * \delta_y^2/4) * exp(-f(P_i, C_x) * \delta_x^2/2)
\end{aligned} \tag{12}
$$

with $\delta_y = f(P_i, C_x)/f(P_i, C_x) * (1 - \delta_x) - 1$.

Using derivation we find the values of δ_x and δ_y which minimize the above probability: $\delta_x = -\frac{f(P_i,C_y) - f(P_i,C_x)}{2*f(P_i,C_y) + f(P_i,C_x)}$ and $\delta_y = \frac{2*f(P_i,C_x) - 2*f(P_i,C_y)}{2*f(P_i,C_y) + f(P_i,C_x)}$. The minimized probability is then:

$$
\begin{aligned}
&Pr_{min}\Big[|S_y| > |S_x|\Big] < \\
&\quad exp\left(-f(P_i, C_y) * \left(\frac{2 * f(P_i, C_x) - 2 * f(P_i, C_y)}{2 * f(P_i, C_y) + f(P_i, C_x)}\right)^2 /4\right) * \\
&\quad exp\left(-f(P_i, C_x) * \left(\frac{f(P_i, C_x) - f(P_i, C_y)}{2 * f(P_i, C_y) + f(P_i, C_x)}\right)^2 /2\right)
\end{aligned}
$$

which gives:

$$
\begin{aligned}
&Pr_{max}\Big[|P_i \cap C_x| > |P_i \cap C_y|\Big] > 1- \\
&\quad exp\left(-f(P_i, C_y) * \left(\frac{2 * f(P_i, C_x) - 2 * f(P_i, C_y)}{2 * f(P_i, C_y) + f(P_i, C_x)}\right)^2 /4\right) * \\
&\quad exp\left(-f(P_i, C_x) * \left(\frac{f(P_i, C_x) - f(P_i, C_y)}{2 * f(P_i, C_y) + f(P_i, C_x)}\right)^2 /2\right)
\end{aligned}
$$

$\blacksquare$

Proof of Theorem 5.1. In the basic PCIR approach, each peer selects and joins exactly one peer group. Let p denote a peer, and g a group of peers. With $DC(\cdot)$ we denote the document collection of a peer or a group. Super peers form the document collection of their group by concatenating the document collection of all peers in the group as follows: $DC(g) := \bigcup_{\forall peer:p \in g} DC(p)$.

The dictionary size of the group collection follows Heap's law. $D_g \approx k * len(DC(g))^\beta$ where k and β are collection-characteristic values. The dictionary size of a peer $p \in g$ also follows Heap's law: $D_p \approx k * len(DC(p))^\beta$. Then, the expected ratio $E\left(\frac{D_p}{D_g}\right)$ is:

$$
E\left(\frac{D_p}{D_g}\right) = \frac{k * len(DC(g))^\beta}{k * len(DC(p))^\beta}
$$

Each group holds on average n/n_{sp} peers. Thus, the average collection length per group equals to $len(DC(p) = n/n_{sp} * len(DC(p)$, and:

$$
E\left(\frac{D_p}{D_g}\right) = \frac{len(DC(p))^\beta}{(n/n_{sp} * len(DC(p)))^\beta} = \left(n_{sp}/n\right)^\beta \tag{13}
$$

We simplify the cost expression for the basic PCIR approach ($C_{basic} = n_{sp} * D_g * (\log(n) + 1)$) as follows: the term $n_{sp} * D_g * \log(n)$ is the dominant term in the cost equation for the basic PCIR (Equation 5), and closely approximates the total cost. This gives:

$$
E\left(\frac{C_{basic}}{C_{flat}}\right) \approx \frac{n_{sp} * D_g * (\log(n) + 1)}{n * D_p * (\log(n) + 1)} \tag{14}
$$

From equations 13 and 14 we get:

$$
E\left(\frac{C_{basic}}{C_{flat}}\right) \approx \frac{n_{sp} * (n/n_{sp})^\beta}{n} = \left(n_{sp}/n\right)^{1-\beta} \tag{15}
$$

$\blacksquare$