Approximate algorithms for efficient indexing, clustering, and classification in Peer-to-peer networks

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades
Doktor der Naturwissenschaften

Dr. rer. nat.

genehmigte Dissertation von

M.Sc. Odysseas Papapetrou

geboren am 18. April 1978 in Nicosia, Zypern

Referent	Prof. Dr. Wolfgang Nejdl
Ko-Referent	Prof. Dr. Norbert Fuhr
Tag der Promotion	18.04.2011

Keywords	Peer-to-peer Networks Information Retrieval Data Mining
Schlagworte	Peer-to-peer Netze Information Retrieval Data Mining

Acknowledgments

Conducting research for a doctorate inevitably requires a huge amount of effort. The never-failing advice is simple: *work hard, collaborate, and get frequent feedback from people you trust*. I was fortunate to have a large set of such persons, to ask for advice, collaborate, and discuss with them, during my studies; friends, colleagues, and family, and the following list is by no means exhaustive.

I owe a lot to my wife, colleague, and frequent coauthor, Ekaterini Ioannou, for her patience and support during my study. Her skills, support, and advices, with respect to both personal and academic problems were very important and helpful for my endeavor.

Professor Wolfgang Nejdl, my advisor, had a principal role in my studies. His continuous support during my studies and his success on maintaining a good atmosphere inside L3S, made the PhD process an unforgettable experience. My mentor, friend, and coauthor, Dr. Wolf Siberski, also supported me during my PhD studies. His patience and experience were very important and helpful for me.

Being in an institute of the size of L3S gave me the opportunity to collaborate with many great colleagues. Dr. Dimitrios Skoutas stands out, for his contribution and support. Dr. Skoutas spent a significant amount of time to read many of my manuscripts and provide comments, but also to discuss new ideas and provide feedback. Professor Wolf-Tilo Balke, now at the University of Braunschweig, was also always there for me, to offer advice and to provide solutions, with his own distinctive, highly appreciated, way. I was also fortunate to collaborate with Professor Norbert Fuhr, from University of Duisburg-Essen. I learned a lot from this collaboration, and I consider this to be a great experience for me.

I greatly and unconditionally recommend collaboration with all these persons to everybody, given such a chance.

Zusammenfassung

Peer-to-Peer-Netze haben die Art und Weise revolutioniert, in der Inhalte geteilt werden, kommuniziert wird, und komplexe Berechnungsaufgaben verteilt durchgefhrt werden. Etwa zehn Jahre nach ihrer Einführung kann man P2P-Technologie in vielen kommerziellen Anwendungen finden, z.B., im Kontext von Internet-Telefonie oder in Netzen zur Verteilung von Dateien.

In dieser Dissertation beschäftigen wir uns mit den folgenden Herausforderungen, die in P2P-Netzwerken für Information Retrieval immer wieder vorkommen:

- Wir zeigen, wie ein verteilter invertierter Index für die Beantwortung von Stichwort-Anfragen effizient gepflegt werden kann. Unser Ansatz bertrifft hinsichtlich der Effizienz bisherige Verfahren um eine Größenordnung.
- Wir stellen einen Algorithmus für die effiziente Erkennung von Near-Duplicates vor, der sich sowohl für Text als auch Multimedia-Inhalte eignet. Wir zeigen, wie sich dieser Algorithmus an Systemanforderungen und Netzwerkcharacteristika anpasst, um die Kosten zu minimieren, und weisen einen Performancegewinn gegenüber früheren Ansätzen nach, der häufig eine Größenordnung übersteigt.
- Wir schlagen ein skalierbares Verfahren für Text Clustering in P2P-Netzwerken vor. Dieses Verfahren nutzt die für Texte typische Term-Verteilung aus, um die Kosten des Clustering erheblich zu reduzieren und adressiert so die Beschränkungen bisheriger Ansätze bezüglich der Qualität, Effizienz und Skalierbarkeit.
- Wir beschreiben einen neuartigen Ansatz für kollaborative Klassifizierung in einem P2P-Netzwerk, der es den teilnehmenden Nutzern ermöglicht, ihre Klassifikationen zu verbessern, indem sie kompakte Repräsentationen ihrer lokalen Modelle austauschen. Im Gegensatz zu bisherigen Verfahren beruht der neue nicht auf dem Austausch persönlicher Daten und Dokumente, und ist damit ein erheblicher Fortschritt in Bezug auf Datenschutz- und Urheberrechts-Aspekte, während er gleichzeitig problemlos für große Netze skaliert.

Alle vorgeschlagenen Algorithmen ermöglichen eine genaue Steuerung des Tradeoffs zwischen übertragungskosten und Qualität der Approximation, und sind daher für eine große Bandbreite von Anwendungen und Einsatzszenarios geeignet. In dieser Arbeit wird ihre Effizienz und Effektivität sowohl wird durch umfangreiche experimentelle Untersuchungen als auch durch theoretische Analysen nachgewiesen.

Abstract

Peer-to-peer networks have revolutionized the way we share information, the way we communicate, and the way we distribute the computation of difficult problems. Almost a decade after their introduction, such networks are found behind many commercial successes, such as IP telephony, video streaming, and P2P file sharing applications.

In this dissertation we consider the following challenges, that frequently occur in P2P networks with information retrieval requirements.

- We show how to efficiently maintain a distributed inverted index which can be used for answering keyword queries. Our approach outperforms the state of the art approaches by an order of magnitude.
- We present an algorithm for efficient near duplicate detection, targeted to multimedia and text files. We show how the algorithm adapts to the system requirements and the network properties for minimizing the cost, and demonstrate the performance improvement, often surpassing an order of magnitude.
- We propose a scalable method for text clustering in P2P networks. The method exploits the inherent skew of the term distribution in text documents to drastically reduce the cost of clustering, thereby addressing the limitations of the state of the art approach with respect to quality, efficiency, and scalability.
- We describe a novel collaborative classification method constructed over a P2P network, which enables the participating users to enhance their classifiers by exchanging reduced, highly compact classification models. In sharp contrast to earlier approaches, the method does not require exchange of raw private data, addressing privacy and copyright concerns, and scales to large networks.

All proposed algorithms enable controlling the tradeoff between network cost and approximation quality, and are thereby suitable for a wide range of application scenarios and deployment setups. Their efficiency and effectiveness are evaluated through extensive experimental setups, using real-world data, and through theoretical analysis.

Contents

Chapter 1

Introduction

1.1 Motivation

The Peer-to-peer (P2P) model has attracted the attention of both the Databases and Information Retrieval communities in the past decade, as well as the users' interest. Their promise for infinite scale, and the recent advances in networking allowing for home PCs to have a cheap, stable, and fast connection to the Internet, made the model appealing for several applications. A recent study showed that P2P applications account for the majority of the network traffic, reaching up to 83% in Eastern Europe countries [147]. Accordingly, the research literature is rich with papers investigating how P2P networks can be efficiently constructed and maintained, as well as with proposals suggesting new application scenarios.

Among the frequently considered application scenarios are the ones involving sharing of resources, such as computational and network resources, storage space, and data. For example, the BOINC framework[1], the powering framework behind SETI@home and Einstein@home projects, enables sharing of computational resources over a P2P network for solving otherwise intractable problems. SummaryCache enables sharing network and storage space, for constructing a distributed caching system. Skype and Veoh exploit sharing of network resources over P2P networks, to enable telephony and streaming video. pStore [13] and PeerStore [88] propose a P2P backup network, where each peer contributes disk space to store encrypted data of other users, whereas Microsoft's FarSite [22] project focuses on sharing storage space to construct a distributed file system. Minerva [16] and Alvis [154], as well as commercial P2P file sharing systems such as Limewire, focus instead on sharing files, such as documents, videos, and audio files.

P2P file sharing was the first commercially exploited P2P application, and immediately gained the critical mass of users to succeed. Its popularity also triggered substantial research efforts toward scalable information retrieval (IR) methods, since existing centralized methods could not efficiently cope with the inherent file distribution in the new paradigm. As a result, in the course of the last decade, several challenges were addressed and novel P2P information retrieval applications were proposed.

The focus of this dissertation is also on information retrieval over P2P file sharing networks. We advance the state of the art in the area by contributing a set of essential components for scalable and advanced information retrieval. We focus on four fre-

[1] http://boinc.berkeley.edu/

quent problems: (a) constructing and maintaining a distributed inverted index over a Distributed Hash Table, (b) near duplicate detection, (c) text clustering, and, (d) text classification.

Distributed inverted indexes constructed over Distributed Hash Tables (DHTs) are arguably one of the most fundamental and frequently used components in P2P systems. They are frequently employed to enable information retrieval in large-scale P2P systems, where central solutions face scalability constraints. Minerva [16], Alvis [103], and PIER [74] are only few of the examples relying on DHTs to enable P2P information retrieval. DHTs provide the necessary coordination and indexing substrate for distributed wikis [128], web caches [76], data stores [85], domain name systems [34], and spontaneous social networks [108, 107]. Owing to their ability to handle churn, they serve as an indexing layer for popular file sharing networks, such as Bittorrent, Limewire, and Kad. However, as we show later, and as also reported by other researchers, e.g., Li et al. [94], maintaining the index becomes very inefficient when considering textual data. Therefore, there is a need for further research towards increasing the performance and scalability of inverted indexes constructed over DHTs.

Near duplicate detection is also important in the context of information retrieval, and has been already considered and even integrated in commercial web search engines as a mechanism to improve IR quality [187, 78, 180]. Frequent application scenarios for near duplicate detection in the P2P domain include finding different versions or encodings of the same resource that might be better suited for the individual user device, parallelizing the downloading process of large multimedia files, detecting resources that violate the copyright laws, and removing duplicates to save disk space [177, 171, 188, 51, 175]. To enable scaling up to large networks, it is important that the near duplicate mechanism is efficient, fully decentralized, and avoids bottlenecks. Hence, this dissertation also investigates how to increase the query throughput and performance of P2P near duplicate detection beyond the state of the art methods, and to enable scaling up to large real-world P2P networks with massive data.

Clustering is another well-established technique, frequently considered in the context of P2P information retrieval, for instance as an alternative methodology to navigate through a large volume of information, or as part of advanced information retrieval algorithms [101, 176, 83]. Further applications of P2P clustering include constructing semantic overlay networks [166, 138], enabling cross-domain multi-organizational clustering of data [40], and outlier detection in distributed streams [11]. However, as we show later, previous P2P clustering algorithms [42, 47, 69, 73], including the state of the art, have significant scalability constraints, and cannot cope with text data. Consequently, further research is required towards a P2P text clustering algorithm that scales to large networks and yields high quality solutions.

Similar to clustering, classification is also frequently used, either stand-alone for information organization or as a useful enhancement in information retrieval. Frequent scenarios include classification between spam and ham for emails, between personal and public for photos, and between relevant and not relevant with a query for documents. Collaborative classification, where users share information to enhance their local classifiers, has revolutionized the email spam filtering field, and ended up in several commercial industrial-strength spam filters, like Google Gmail, Yahoo, and Hotmail. Collaborative classification was also explored in the context of P2P, e.g., [165, 102, 152, 6]. In this work, we propose a novel P2P collaborative classification approach, which outperforms existing approaches with respect to classification accuracy, requires negligible network cost, and scales to large networks and to high-dimensional data, such as text.

1.2 Contribution

This thesis advances the state of the art in the following building blocks of P2P information retrieval.

I. Building the inverted index over a DHT. As already mentioned, the prevalent approach for enabling P2P information retrieval is by employing an inverted index, constructed over a distributed hash table. This involves two challenges: (a) maintaining the distributed inverted index, and, (b) using this index to locate and rank the relevant results. The second challenge has been successfully addressed by a plethora of approaches, e.g., [3, 127, 103, 16, 126, 82, 141]. However, it has been shown that the maintenance of the distributed inverted index is very inefficient and imposes scalability limitations, especially for indexing textual data using the standard 'bag of words' information retrieval model [94]. Towards alleviating this problem, several approaches were proposed [151, 36, 100], but these either require extensive additional work from the users for manually selecting good indexing features, or have a negative impact on the information retrieval quality.

To alleviate the scalability constraints without sacrificing the completeness of the index we propose PCIR, which drastically reduces the indexing cost compared to earlier approaches. PCIR pertains the 'bag of words model', and addresses the scalability issue by enhancing the indexing workflow with an intermediary step. Peers exploit peer clustering and Bloom filters to form small groups, and coordinate the index maintenance process within each group. This allows exploiting the high term overlap between peers, and yields significant performance enhancements. The approach does not depend on a particular family of relevance functions, or a particular algorithm for query execution. It can be combined with the majority of existing P2P IR systems to optimize the network usage, without modifying the way queries are executed, or altering the quality of the results. We demonstrate experimentally the benefits of PCIR by applying it to four different state of the art P2P Information retrieval networks. PCIR yields an order of magnitude less network cost compared to competitive algorithms without negatively affecting the quality and completeness of the index.

The PCIR algorithm was described in the following publications:

- Odysseas Papapetrou, Wolf Siberski, Wolfgang Nejdl. PCIR: Combining DHTs and Peer Clusters for Efficient Full-text P2P Indexing, Computer Networks 54(12): 2019-2040 (2010), Elsevier.

- Odysseas Papapetrou, Wolf Siberski, Wolfgang Nejdl. Cardinality estimation and dynamic length adaptation for Bloom filters, Distributed and Parallel Databases 28(2):119-156 (2010), Springer.

- Odysseas Papapetrou. Full-text Indexing and Information Retrieval in P2P systems, in: Proc. Extending Database Technology PhD Workshop (EDBT), 2008, Nantes, France.

- Odysseas Papapetrou, Wolf Siberski, Wolf-Tilo Balke, Wolfgang Nejdl. DHTs over Peer Clusters for Distributed Information Retrieval, in: Proc. IEEE 21st International Conference on Advanced Information Networking and Applications (AINA), 2007, Niagara Falls, Canada.

II. Near duplicate detection. Several approaches for P2P near duplicate detection have been already proposed. However, as shown later, without careful manual configuration from the user and adaptation of the algorithm to the contents of the network, the existing P2P near duplicate detection methods, e.g., [14, 68, 51, 180] may have a substantially higher cost than the optimal, with the difference often reaching to several orders of magnitude compared to the minimal cost. Therefore, we require a P2P near duplicate detection algorithm that can optimize its configuration according to the user requirements and network properties, to minimize the network cost. We propose POND, a P2P method based on Locality Sensitive Hashing, which enables this adaptation and minimizes the network cost. POND allows users to select a tradeoff between cost and completeness of the results, expressed using probabilistic guarantees, and optimizes the system configuration accordingly, so that the network cost is minimized for the desired probabilistic guarantees. The good properties of POND are demonstrated experimentally, with simulations of up to 100,000 peers, and using three real-world datasets of text, video, and audio files, with a total size of more than 200 GBytes. The experimental results show that performance improvements of several orders of magnitude are feasible, without a negative impact on quality.

Parts of this work have appeared in:

- Odysseas Papapetrou, Sukriti Ramesh, Stefan Siersdorfer, Wolfgang Nejdl. Optimizing Near Duplicate Detection for P2P Networks, in: Proc. IEEE International Conference on Peer-to-Peer Computing (P2P), 2010, Delft, Netherlands.

III. P2P clustering. Previous P2P text clustering methods, such as [42, 47, 69, 73], run into scalability issues for large networks, or rely on a particular assumption for the distribution of the text files to the participating peers, which is incompatible with real-world scenarios where each peer carries its own documents. To address these constraints we propose PCP2P, a P2P text clustering algorithm with probabilistic guarantees. The algorithm is designed with special concern on scalability and network efficiency, and is suitable for clustering high-dimensional and skewed data like text documents. Through an extensive theoretical analysis we derive probabilistic guarantees for the clustering quality of PCP2P, and show how these are used to configure the desired cost/quality tradeoff. We demonstrate the efficiency and effectiveness of the algorithm through a large scale experimental evaluation, and an extensive comparison with the state of the art methods.

PCP2P has been partially presented in:

- Odysseas Papapetrou, Wolf Siberski, Norbert Fuhr. Decentralized Probabilistic Text Clustering, under revision at TKDE, 2010.
- Odysseas Papapetrou, Wolf Siberski, Norbert Fuhr. Text Clustering for Peer-to-Peer Networks with Probabilistic Guarantees, in: Proc. 32nd European Conference on Information Retrieval (ECIR), 2010, Milton Keynes, UK.
- Odysseas Papapetrou. Full-text Indexing and Information Retrieval in P2P systems, in: Proc. Extending Database Technology PhD Workshop (EDBT), 2008, Nantes, France.
- Odysseas Papapetrou, Wolf Siberski, Fabian Leitritz, Wolfgang Nejdl. Exploiting Distribution Skew for Scalable P2P Text Clustering Databases, in: Proc. Information Systems and Peer-to-Peer Computing (DBISP2P) 2008, Auckland, New Zealand.

IV. P2P classification. Several P2P collaborative classification algorithms were proposed [165, 102, 152, 6], which enable users to share information over a P2P network, and enhance their local classifiers. However, as we demonstrate later, these algorithms have significant scalability limitations, and do not perform well on text data. Furthermore, most of these algorithms rely on sharing of raw training and testing data, such as emails, which imposes serious privacy and copyright concerns, and limits their applicability. To address these issues we propose CSVM, a collaborative classification framework constructed over a P2P network. The method focuses on text classification, and addresses the limitations of the state of the art systems concerning communication cost, privacy, and scalability. Peers participating in CSVM share information enabling them to improve their classifiers to a quality which can otherwise be accomplished only by manual user intervention, i.e., more extensive training. The algorithm allows the participating peers to configure the tradeoff between cost and quality. Comprehensive experimental evaluation using several massive real-world datasets demonstrates that CSVM outperforms the state of the art collaborative classification methods with respect to effectiveness, and closely approximates the quality of a centralized solution with only a small fraction of its computational and network cost, which is easily affordable even by mobile devices.

Different aspects of CSVM are described in the following publications:

- Odysseas Papapetrou, Wolf Siberski, Stefan Siersdorfer. Collaborative Classification over P2P networks, in Proc. WWW (Companion Volume), 2011.
- Odysseas Papapetrou, Wolf Siberski, Stefan Siersdorfer. Efficient Model Sharing for Collaborative Text Classification, Technical report, 2011.

1.3 Structure of the Dissertation

The next chapter provides the necessary background for this work. In Chapter 3 we describe PCIR, our proposal for scalable P2P indexing, and demonstrate how it can be applied to existing P2P information retrieval systems for reducing the network cost. Chapter 4 describes POND, the near duplicate detection system, and demonstrates its efficiency and effectiveness with comprehensive experiments using up to 200 GBytes of text files, videos, and audio files. In Chapter 5 we describe PCP2P, a P2P text clustering algorithm designed with special concerns on scalability and network efficiency. We derive probabilistic guarantees for the quality of PCP2P, and demonstrate its efficiency and effectiveness through a large-scale experimental evaluation. Chapter 6 introduces CSVM, a text classification algorithm that scales over P2P networks, and presents comprehensive experimental results with several massive real-world data collections, and comparisons with the state of the art algorithms. Finally, in Chapter 7, we discuss future work, and conclude the thesis.

Chapter 2

Foundations

Peer-to-peer topologies can be broadly classified into four types: (a) P2P with central servers, (b) unstructured P2P, (c) super peer networks, and, (d) structured P2P. We now provide an overview of these topologies.

2.1 P2P with Central Servers

P2P with dedicated central servers was first popularized by Napster (1999-2001), a file sharing network focused on sharing music files. The network is built around a central server hosting an inverted index. Peers join the network by contacting this server and registering their resources. The location of each resource is indexed using the extracted resource features, such as the filename and meta data for multimedia files, or the terms for text documents. Query execution also depends on the central server. All queries are forwarded to the central server, and the query keywords are looked up in the inverted index, to find the relevant results. The results are returned to the query initiator, which then decides which of the resources should be retrieved, and contacts their corresponding peers directly for retrieving the resources.

The BitTorrent protocol also belongs to this category, as it relies on dedicated servers for indexing the resources. The key idea behind the protocol is that each resource is described using a small descriptor file, called torrent. Torrents are distributed by conventional means, usually uploaded to dedicated web servers, and can be located using specialized search engines. Each torrent is also associated with a special peer, called a tracker, responsible for keeping track of the peers that own and can provide the resource. For downloading a resource, the user first locates the corresponding torrent and the associated tracker. The tracker coordinates the downloading process by providing a list of peers that can provide parts of the resource.

This topology scales better than conventional information sharing approaches, web hosting for example, since the central server does not need to maintain copies of all resources locally. It only needs to keep an index of the resource locations, using a limited number of features per resource, for instance, the filenames in Napster, or the torrents in BitTorrent. However, as the resources, peers, and features per resource increase, this topology also runs into bottlenecks. An increased query rate further aggravates the workload of the server, since all queries need to be executed from the same server. Finally, the high network cost for providing the central server, undertaken by a single party, combined with the high churn rate expected in P2P networks, make

this architecture economically unattractive.

A non-technical concern of these server-centric P2P networks, which also contributed towards the disfavor of the model, is that the server owners are considered legally responsible for the search functionality. Due to the frequency of copyright infringements in these networks, several companies, including Napster, suffered frequent legal consequences for providing the search services.

2.2 Unstructured P2P Networks

Realizing the limitations of the central-server architectures, the decentralized unstructured P2P topology quickly emerged. The distinguishing characteristic of this topology is that the participating peers maintain links to a few selected neighbors, which enable information sharing without the need of central servers. A new peer joins the network by locating a set of old peers, usually using a central registry or some well-known IP addresses, and establishing a link with these peers. Concerning query execution, different approaches were proposed in the literature. The simplest approach is based on query flooding. The query initiator forwards to all its neighbors the query, annotated with a Time-To-Live value. Each neighbor answers the query by sending its local results to the query initiator, and forwards the query to its own neighbors, which repeat the process recursively until the Time-To-Live expires. Variants of this topology are used in Freenet [30] and Gnutella [162], two popular P2P file sharing networks.

To reduce the cost of flooding and to speed-up the query execution process, other unstructured architectures were also proposed. An architecture which also found application in recent Gnutella versions bases query routing in peer synopses, e.g., [35, 178, 86, 37]. Peers *gossip* summaries of their contents periodically, using synopses, such as Bloom filters. These synopses are used during querying, to route the query only to the peers most likely having related documents. Another influential architecture which improves query performance is based on a semantic overlay network, where semantically related peers are clustered together, e.g., in [36, 98, 45]. In addition, several mathematical models were proposed for the network construction and the query routing phase that increase the probability that each query retrieves all relevant results, and improve the scalability of such networks [28, 80, 183, 104, 114].

The main benefits of these approaches compared to the central-servers topology is that they are more scalable, they evenly distribute the network and computational cost to all participating peers, and do not suffer from a single point of failure. This makes unstructured P2P networks an appealing solution for several application scenarios, such as video streaming [167], privacy and anonymity [30], and collaborative classification presented in Chapter 6. These networks also formed the basis of several popular file sharing P2P networks, the most famous being Limewire and Freenet. However, due to the absence of an inverted index, unstructured networks are not effective at locating unpopular resources, or, in the words of Rodrigues and Druschel [144], they are good at finding “hay”, but not at finding “needles”. Furthermore, their network efficiency and performance is still limited, due to the extensive gossiping required for exchanging the synopses, and the flooding during query processing [143].

2.3 Super Peer Networks

To balance between the efficiency of P2P networks with central servers, and the scalability and robustness of unstructured architectures, super peer topologies were proposed, for instance in [122, 179, 125, 136, 96]. The key aspect behind these topologies is that some of the peers have higher capabilities than others, such as faster network connection and a higher uptime, which could be exploited for increasing the stability and efficiency of the network. Therefore, super peer topologies construct a network with two types of peers, the ordinary peers, and the more powerful super peers. Only the super peers participate in the unstructured network. The ordinary peers instead find and join a super peer, through which they have access to the network. Upon joining, ordinary peers send their synopses to their super peers, which act on their behalf for query processing. In particular, the query initiator forwards its query to its super peer, which first looks up its collected synopses. If the query can already be answered, the super peer returns the locations of the relevant resources to the query initiator. Otherwise, the super peer floods the query to its neighboring super peers, which recursively answer the query.

Several extensions were also proposed to address limitations of the basic topology. For instance, Yang and Garcia-Molina [179] describe how redundancy at the super peer level increases robustness, and enables load balancing. Kleis et al. [81] take into account the underlying network conditions when constructing the super peer network, thereby yielding better performance and scalability. Garbacki et al. [64] consider the semantic similarity and interests of the peers for constructing the network. This yields an increase in the locality of the queries, and makes query processing more efficient.

2.4 Structured P2P Networks

This is the prevailing P2P topology. It offers efficiency and effectiveness guarantees, and is a central component of several popular file sharing networks, e.g., Limewire and BitTorrent DHT. Our contributions also employs this topology to a large extend. The common principle behind structured P2P networks is an overlay structure called Distributed Hash Table (DHT), which enables mapping of peer and resource identifiers to a common address space, allowing for direct allocation of resource identifiers to peers. DHTs follow the abstraction of hash tables, offering a *put(key, value)* method, which associates *value* with *key*, and a *get(key)* method, which returns the *value* associated with *key*. Typically, these methods have network complexity of $O(\log(n))$ messages, where n is the number of participating peers.

There is a plethora of DHT variants in the literature, for instance, Chord [158], P-Grid [1], Pastry [145], Viceroy [106], CAN [140], and Kademlia [111]. We now demonstrate the basic DHT functionality, using Chord as example. Figure 2.1 presents a sample DHT. Each peer is assigned a unique identifier in the range of $[0, 2^r - 1]$, where r is a system parameter. For instance, most Chord implementations set r to 160 (in the figure we use $r = 6$, for illustration purposes). This places the peers in a conceptual ring. The significance of this placement is that each peer undertakes responsibility for maintaining all keys in the range of its own identifier and the identifier of the previous peer in the ring (e.g., P_6 is responsible for all keys between 2 and 6). Then, each peer constructs its finger table, by establishing pointers to other peers. The peers included in this table are selected such that their keys have an exponentially increasing distance with the source peer. For example, P_6 will establish pointers to the peers responsible

for storing the keys $6 + 2^i$, for $i = [1 \ldots 5]$.

This structure enables key lookups with logarithmic complexity. To execute a DHT lookup for a query q, the query initiator peer p_q first uses a predefined hash function for computing hash(q), a hash value of q in the range of $[0, 2^r - 1]$. Cryptographic hash functions such as SHA-1 and MD5 are usually employed for this purpose. Then, p_q selects from its finger table the peer p' with the identifier satisfying $\text{id}(p') \geq \text{hash}(q)$ and $minarg_{p'}(\text{hash}(q) - \text{id}(p'))$. Peer p_q forwards the lookup message to p'. If p' is the responsible peer for that key, it answers back to p_q with the value registered for q. Otherwise, p' repeats the routing process, transparently from p_q, and forwards the lookup message to the best peer candidate from its own finger table. The process is repeated until the responsible peer for the query is found, which responds back to p_q directly with the answer. It can be shown that the lookup process scales logarithmically with the number of peers.

Apart from the basic lookup functionality, several other issues were considered in the context of DHTs. To recover for churn, e.g., peers leaving unannounced due to network failures, Chord replicates each peer's DHT fraction to a number of its successors. When a peer fails, its successors take over the *orphaned* range, and no data is lost. A similar approach is followed by Pastry [145]. Generally, by controlling the replication factor, and under the assumption that there is no correlation between the peer identifiers and the peer's geographic location, ownership, or network connectivity, DHTs provide guarantees for the completeness and connectivity of the graph. Rhea et al. [142], and Li et al. [95] elaborate further on the topic.

DHTs have also been extended to handle high-dimensional data. For instance, the CAN DHT [140] considers a d-dimensional space split into n zones, each assigned to a peer. Peers construct their finger tables by including all peers responsible for neighboring zones, i.e., zones that have $d - 1$ overlapping dimensions. A simple greedy algorithm is employed for lookups, with network complexity of $O(d \times n^{1/d})$ messages. Mercury [19] and HotRoD [134] handle multi-attribute data by constructing a single virtual ring for each dimension. For DHT lookup, only the virtual rings with dimensions relevant to the query are looked up.

Several DHT extensions were also proposed to address specific DHT limitations. To address the problem that DHT lookups may impose high latency due to the geo-

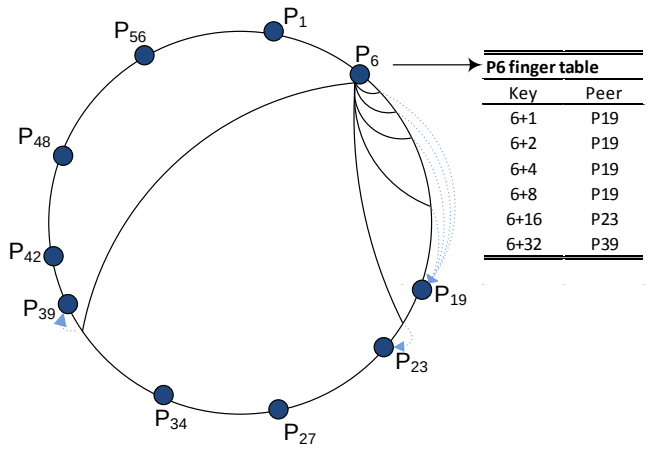

Figure 2.1: Chord ring

graphic distance of the involved peers, Pastry DHT takes into account network locality. Due to the replication imposed for handling churn, each lookup can be served from k adjacent peers. Therefore, instead of always forwarding each message to the peer with identifier closest to the key, the k closest peers with the key are detected from the finger table, and the message is forwarded to the one minimizing the network distance.

To address the large number of messages created in DHT networks, which can overload the network and cause problems to weak participants, Klemm et al. [82] consider buffering of the DHT messages. Particularly, small DHT lookup messages are buffered, and aggregated to larger messages with the same destination. This imposes a slight additional latency on the DHT lookups and increases the failure rate of DHT lookups, but it also reduces the network load substantially.

The problem of load balancing for peers holding popular keys has also been considered extensively, e.g., [67, 135, 134]. The frequent approach [67, 189] is to change the region associated with each peer, such that more peers end up sharing the crowded regions. For this, the notion of virtual servers is employed. Each peer hosts several virtual servers, covering the responsibility range of the peer. When a peer becomes overloaded it reassigns some of its virtual servers to neighboring peers. Network proximity is also considered for choosing where each virtual server is reassigned, efficiently computed via landmarking [189]. Pitoura et al. [134] follow a different approach, forming virtual DHT rings over the same DHT overlay to maintain the data. The rings are virtual, in the sense that they do not require materialization, with additional finger tables. The data replication imposed by this approach also enables recovery from peer churn with no data loss. Bharambe et al. [19] address load balancing differently. Peers construct a histogram on the popularity of the DHT regions, indicating the expected load of each peer. The load imbalance between peers is addressed by moving peers from the unpopular DHT regions to the popular DHT regions, to assist the overloaded peers.

A shortcoming of traditional DHTs is that they only enable exact matching. Several proposals address this issue, by enabling range queries over P2P [134, 135, 186, 19, 63]. For example, Gao et al. [63], and Zheng et al. [186] propose layered DHT structures, resembling a range tree and a segment tree respectively, which naturally enable range queries. Similarly, P-Grid supports range queries by nature due to its trie backbone. Pitoura et al. [134, 135] follow a different approach, employing locality sensitive hashing to generate the keys, thereby ensuring that similar values will end up in the same or in near-by peers of the DHT ring.

The principal application of DHTs is for constructing a distributed inverted index, for enabling keyword queries over P2P networks, e.g., [16, 154, 103, 74]. Each peer processes its local collection to extract the main features, or terms, and posts these terms in the DHT, with pointers to the relevant documents. For query execution, the query terms are looked up in the DHT, to retrieve the posting lists with the relevant documents. We will discuss the construction and maintenance of distributed inverted indexes in more detail in Chapter 3.

There are also several approaches combining DHTs with other topologies to address particular application challenges. For example, Loo et al. [100] propose a combination of DHTs and unstructured networks in the context of keyword search; unstructured networks are used for locating the popular resources, whereas a DHT index is used to answer the rare queries. In the context of this thesis, we combine DHTs and super peers to increase the efficiency of the inverted index maintenance, and to enable distributed clustering. We discuss the choices involved for such a combination in Chapters 3 and 5.

The purpose of this chapter was to briefly introduce the basic P2P topologies, in the depth required for this thesis. A more detailed introduction to P2P systems and P2P information retrieval can be found in several books and articles, e.g., [144, 156, 109]. Furthermore, in the following chapters, we will provide a more focused discussion for some of these approaches, present their limitations, and compare them with our contributions.

Chapter 3

Distributed Indexing for Information Retrieval

Information retrieval is one of the most frequently considered applications for P2P networks. A naive approach to enable P2P information retrieval is to use a devoted powerful node for maintaining an inverted index of the contents of all peers. Considering the high expected churn factor for P2P networks, the large volume of the contained information, and the targeted network size, this approach does not scale. As such, a significant body of work has been targeted to alternative approaches, which do not assume centralized services.

The predominant approach uses DHTs to enable a distribution of the inverted index over all participating peers. Each peer analyzes its own collection, and extracts a set of terms, normally after performing basic stemming and filtering of stopwords. For each extracted term, the peer executes a DHT lookup to locate the responsible peer in the network, and posts there its contact details, with the term and term score. With respect to query processing, the query terms are similarly looked up in the index to find the relevant peers. This approach (in the following called *flat DHT indexing*) has been extensively used in the literature, such as in Minerva [16], Alvis [103], and PIER [74]. The approach works well when the number of terms per peer is limited, even for extremely large networks. However, flat DHT indexing becomes prohibitively expensive for full-text indexing, or when the number of distinct terms per peer is high [94]. When each peer features many 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 each 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 the index maintenance cost, e.g., combining DHTs with unstructured networks [100], or indexing only selected terms [151]. As we show in Section 3.2, 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.

In sharp contrast to previous works, we propose an approach which reduces indexing costs significantly without affecting querying cost or the quality of the results. Our approach is called PCIR, short for *Peer Clusters for Information Retrieval*, and is based on a hybrid super peer/DHT topology: we organize peers into groups, each of them rep-

resented 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, as well as the total number of messages is drastically 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 regular peers in the flat DHT approach.

In its basic form, PCIR relies on arbitrary assignment of peers to groups, without taking the peer contents into account. We achieve further improvements by introducing a distributed clustering scheme, such that peers form groups/clusters based on content similarity. This further 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 next section shows how inverted indexes are currently constructed over DHTs, and how the resulting index structure is used to evaluate keyword queries. In Section 3.2 we discuss related work. The basic algorithm and the building blocks of our topology are presented in Section 3.3, whereas in Section 3.4 we introduce the advanced scheme that employs peer clustering to further reduce the network overhead. A cost analysis for both approaches is presented in Section 3.5, and Section 3.6 contains the results of our experimental evaluation. Section 3.7 summarizes the contribution and concludes the chapter.

3.1 Prerequisites

In this section we describe the underlying techniques and the standard approach for DHT-based information retrieval in P2P networks.

3.1.1 DHT-based Inverted Indexes

Our approach relies on the functionality of a Distributed Hash Table to construct an inverted index (cf. Section 2.4). We use Chord as a DHT overlay, but PCIR does not rely on a specific DHT approach; other DHT overlays, such as P-Grid [1] and Pastry [145], could be employed as well.

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 the terms with stopword filtering and stemming as usual (see Peer P_4 in Fig. 3.1 for example). Then, it creates an index entry for each discovered term and *posts* 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.

The information contained in the index depends on the scoring functions to be used. 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

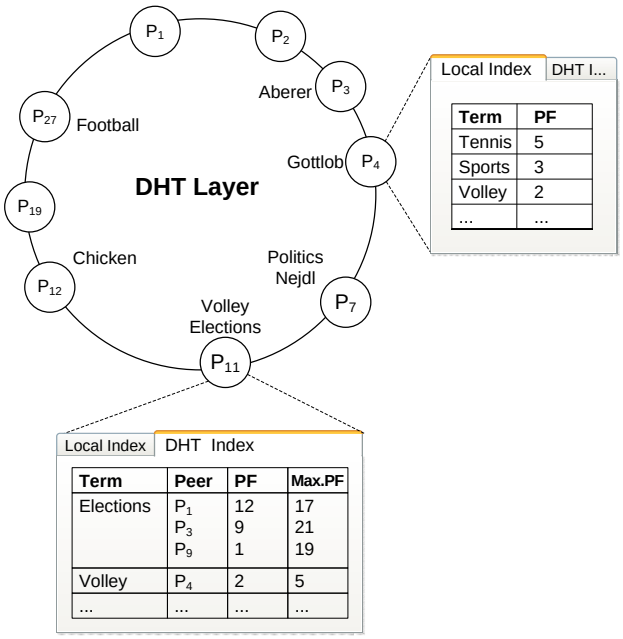

Figure 3.1: Flat DHT topology and index structure

information retrieval systems build and maintain such a document granularity inverted index, e.g., [103]. However, indexing each document individually is expensive in terms of network cost. Therefore, most P2P information retrieval approaches use peer granularity [16], 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 [26]. Additional index variants have also been proposed, which we describe in Section 3.2.

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 $max\_pf(p_i) = \max_{t \in DC(p_i)} pf(t, p_i)$, used for normalization of the peer scores with the size of each peer's collection (see, for example, Fig. 3.1). 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 employed by the state of the art 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 is to attach an expiration time to each index entry. Peers periodically republish their content such that their index entries are renewed 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 indexes form an excellent foundation for distributed query processing, their creation and maintenance incurs a very high number of network messages. Even though 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 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.

3.1.2 DHT-based Query Processing

The work described in this chapter focuses on efficiently maintaining the required information in the distributed inverted index, and it is applicable to all query execution techniques that rely on a DHT-based index. For completeness, we briefly demonstrate the basics of the most frequent technique, where query processing is performed as two-step process [16]. The query initiator first discovers all related peers with the query, 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).

Collection selection. The literature is rich with collection selection techniques for distributed collections. These approaches are directly applicable to P2P systems, since each peer can be considered as a distributed collection. We demonstrate PCIR using CORI [26] for addressing the collection selection problem.

Assume a query Q consisting of terms $\{t_1, t_2, \ldots, t_l\}$. The query initiator looks up each query term in the DHT, and retrieves the corresponding posting lists. 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:

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

with $T_{t,p_i} = d_t + (1 - d_t)\dfrac{\log(pf(t, p_i) + 0.5)}{\log(max\_pf(p_i + 1))}$ and $I_{t,p_i} = \dfrac{\log\left(\frac{n+0.5}{cf(t)}\right)}{\log(n + 1)}$. Parameters d_t and d_b are constants, with a suggested value of 0.4 (for more information on how to set these constants see [26]). $pf(t, p_i)$ 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 (Fig. 3.1) and the data required for CORI score function, we see that all the required data is already stored in the DHT, except of n, the total number of peers. The value of n can be inexpensively estimated with a random walk [110], and periodically redetermined to take churn into account. Thus, for each query term the necessary parameters can be easily retrieved with a single DHT lookup.

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 $tf \times icf$ score. Links to the results are returned to the query initiator, ordered by relevance with the query, and displayed to the user.

3.1.3 Bloom Filters

We use Bloom filters for compactly representing large term sets (Section 3.4). Bloom filters were first proposed in [21], as a space-efficient representation of sets $S = \{e_1, e_2, \ldots, e_n\}$ 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 $F = \{f_1, f_2, \ldots, f_k\}$, 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 $f_i \in F$.

A limitation of Bloom filters is that they do not allow removing of elements. For removing an element, the whole Bloom filter needs to be rebuilt from scratch. A workaround was proposed by Fan et al. [52], called *counting 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, the respective counters are decreased by one. In our context, counting Bloom filters are used to compute the difference between two Bloom filter summaries.

3.2 Related Work

Peer-to-peer information retrieval has been studied in a large number of publications. The first proposals focused on distributed information retrieval in unstructured networks, using approximate system-wide information. PlanetP [37] is one of these systems. It uses gossiping to distribute peer content summaries, encoded as 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 a few dedicated nodes take the responsibility for maintaining the indexes [31, 10]. While these systems scale better compared to the unstructured networks, they require that the super peers have very good network connections and a 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 cost grows logarithmically with network size [3, 103, 16]. As explained in Section 2.4, these systems enable information retrieval by constructing a distributed inverted term index. However, the main weakness of DHT-based P2P IR systems is the high cost for maintaining the inverted index. Existing systems vary the granularity and completeness of the inverted index, targeting different quality/cost tradeoffs. In the ALVIS system [3, 103], peers index the terms for

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

each of their documents individually (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 [16, 17] aggregate their documents' scores per term to produce a peer score for each term. 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 number of DHT lookups per peer is not reduced. In fact, Li et al. [94] 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 3.6, further suggest 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 [61] 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 the size and number of domains, index maintenance can be traded against query efficiency. This approach reduces the number of DHT lookups, but only with important tradeoffs for query execution, either in the quality of the results or in the query cost. When full-text indexing 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. In [151], the peers randomly choose a subset of the terms to be indexed. Crespo and Garcia-Molina [36] 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. [100] 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 compromised.

Nguyen et al. [126] 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, the 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 [154], 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, another query-driven indexing approach called mk-STAT [117] 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.

In Section 3.6 we demonstrate experimentally that PCIR enables substantial network savings also for query-driven indexing. In particular, we apply PCIR on four publishing strategies: two different peer granularity publishing strategies [117, 16], a query-driven strategy [117], as well as a document granularity strategy [141]. The application of our approach to other strategies is straightforward. For example, [137] increases query 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 also these as 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., [126], can also employ PCIR for reducing the network cost, without affecting their query execution part.

Some systems do not index terms at all. For instance, pSearch [160] proposes two alternative dimension models for indexing the documents. The first one, pVSM, is based on the Vector Space model, while the second one, pLSI, reduces the document dimensions using Latent Semantic Indexing. The main problem in both the approaches is the high network cost. In addition, pLSI assigns the LSI computation to a single peer, causing serious bottlenecks and scalability issues for large networks. To aggravate the problem, the proposed load balancing extension [159] induces a huge increase of the network cost. Thus, pSearch does not scale for large networks.

Hierarchical DHTs have also been introduced, as a means to foster efficient bandwidth utilization and a better adaptation to the underlying physical network [60, 191]. 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 [191].

The issue of too many DHT accesses can also be partially alleviated by buffering and aggregating small DHT messages with the same destination [82]. 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.

Super peer based topologies which do not rely on DHTs have also been proposed for Information Retrieval. Cooper [31] proposes to maintain collection-wide information as well as routing tables at designated super peers, called InfoBeacons. Balke et al. [10] collect query statistics at super peers to facilitate efficient term query processing. While these approaches incur significantly less index maintenance effort than DHT-based algorithms, query processing workload is not evenly distributed over the peers, hence impeding scalability.

In addition to the DHT index construction, query processing also introduces several challenges. In [131] we presented a method for increasing the novelty of the results by avoiding duplicate documents. The same problem was studied in [15, 118]. The problem of top-k query execution has been investigated by Balke et al. [10] for super peer networks, but also in [123, 116] for structured P2P networks. In [139] we showed how Bloom filters can be used to construct more efficient query plans, and increase the

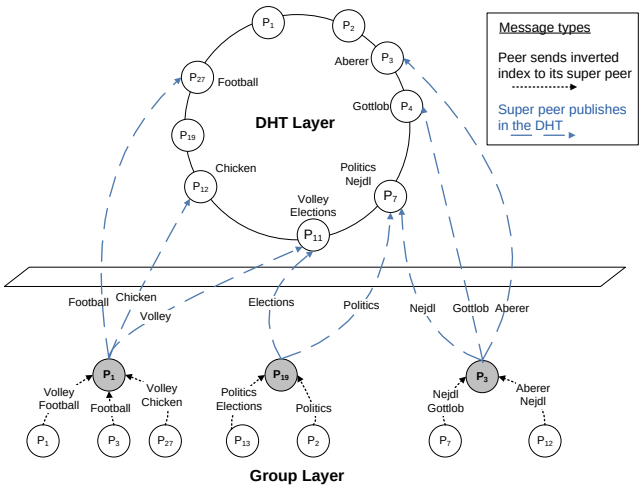

Figure 3.2: The two-layer architecture combines grouping of peers around super peers and an inverted index over a DHT (super peers are gray shaded)

efficiency of the distributed joins required for multi-term queries. These approaches also rely to a DHT infrastructure, and can be combined with PCIR straightforwardly.

3.3 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 compromising the completeness of the inverted index so that existing query execution techniques can be applied, like the one described in Section 3.1.2. 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 exploits this term overlap between peers by forming small groups of peers, and assigning the responsibility for DHT lookups in each group to a selected super peer of the group.

Figure 3.2 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 indexes 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

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 3.3: 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.

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.

The resulting distributed index created from PCIR looks nearly identical to the one constructed by flat DHT Indexing. The only difference is that in addition to the peer information, the super peer adding 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 3.3 shows a sample index. In the following sections we explain how this index is built and maintained in detail.

3.3.1 Peer Life-cycle

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.

Algorithm 3.1 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., 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 specified number of unsuccessful attempts to join an existing peer group. In sharp contrast to other super peer systems, PCIR does not require super peers to be especially powerful, as these are not used during query processing. In fact, as we show later, a super peer in PCIR typically has less network workload than regular peers in flat DHT indexing.

Algorithm 3.1: Peer joining a peer group (basic PCIR)

Input: P_{sp_ratio}: Desired ratio of super peers to peers

```
// a peer joins the network
joinPCIR() {
if decideIfSuperPeer() then
    // become a super peer
    this.isSuperPeer ← true;
    groupInvertedIndex←myInvertedIndex;
else
    // become a normal peer
    this.isSuperPeer ←false;
    // find random super peer
    mySuperPeer ← findRandomSuperPeer();
    // and submit your inverted index
    mySuperPeer.groupInvertedIndex.update (myInvertedIndex);
}

SuperPeer findRandomSuperPeer() {
repeat
    int randomKey ← getRandom();
    peer p ← DHTLookup(randomKey);
    <SuperPeer> listOfSPs← p.allKnownSuperPeers();
    while (listOfSPs.isNotEmpty() & mySuperPeer is null) do
        SuperPeer randomSP← listOfSPs.selectRandom();
        if randomSP.canAcceptMe() then
            mySuperPeer←randomSP;
until SuperPeer found;
return mySuperPeer;
}

boolean decideIfSuperPeer() {
double random← getRandom(0,1);
return random ≤ P_sp_ratio;
}
```

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 the super peers workload 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

Algorithm 3.2: Periodic Peer/Super Peer publishing

```
1  while true do
2      if isSuperPeer then
           // update with my own collection
3          this.groupInvertedIndex.update (myInvertedIndex);
           // publish group's inverted index
4          for term t in groupInvertedIndex do
               // publish peerlist in DHT
5              peer p ← DHTLookup(t.hashValue);
6              p.publish (groupInvertedIndex.getPeerList(t));
7      else
8          if (!mySuperPeer.isAlive () || !mySuperPeer.canAcceptMe ()) then
9              joinPCIR () ;                          // join a new group
10         mySuperPeer.groupInvertedIndex.update (myInvertedIndex);
11     sleep(period);
```

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.

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. 3.2). The publishing process is periodically repeated to compensate churn.

This two-layered publishing process has several advantages compared to flat DHT indexing. Peers can efficiently publish their inverted index at their super peers, since this requires only one message which is easily compressible, and does not generate any DHT lookups. 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. 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 are not required 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. The peers only need to share the same expiration period, and no further synchronization is required.

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 [145]). 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 Algorithm 3.1. Due to super peer churn, the postings of a peer might not be propagated in the DHT within one publishing period. However, the probability that this happens for i consecutive periods is small, and in particular, $pr = c^i$, where c denotes the churn percentage. 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 indexes 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 [38], 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.

3.4 PCIR Clustering-enhanced Algorithm

The 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 group. As a result, super peers of the groups will have fewer distinct terms,

λ	Number of top frequent terms published per super peer
μ	Maximum number of clusters to compare with each virtual peer's Bloom filter
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 3.1: Notations for PCIR

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.

Clustering-enhanced PCIR 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. Peers create their Bloom filters by hashing all their terms in an empty Bloom filter. Each super peer is additionally responsible for maintaining the Bloom filter representation of its cluster's centroid, which equals to the disjunction of the filters of all the peers belonging in 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 discuss the building blocks of clustering-enhanced PCIR in detail. In Section 3.4.1 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 3.4.2 we describe a cluster centroid caching scheme used to alleviate the workload of super peers. Section 3.4.3 describes the required changes at the DHT inverted index for enabling peer clustering. The process of joining peers in the clustering-enhanced PCIR network is put together in Section 3.4.4. In all other aspects, clustering-enhanced PCIR algorithm works like the basic algorithm.

3.4.1 Clustering objective function

The purpose of clustering is to increase the term overlap in the super peers. Therefore, 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 largest overlap. To avoid exchanging large inverted indexes between peers for computing the overlap, peers estimate the cardinality of the overlap using Bloom filters.

The clustering objective function is formally defined as follows:

Definition 3.1. *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:

$$f(P_i, C_x) = \begin{cases} f'(BF_{C_x}, BF_{P_i}) & , \text{if } P_i \notin C_x \\ f'(BF_{\{C_x - P_i\}}, BF_{P_i}) & , \text{if } P_i \in C_x \end{cases} \tag{3.1}$$

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 3.1), and $BF_{\{C_x - P_i\}}$ is the Bloom filter of cluster C_x after removing the contents of 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 their Bloom filters.

Theorem 3.1. *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:*

$$\begin{aligned} E(|A \cap B|) = & \frac{m^2 - m(tb(BF_A) + tb(BF_B)) + tb(BF_A) \times tb(BF_B)}{k \times \ln(1 - 1/m)} \\ & - \frac{m^2 - m(tb(BF_A) + tb(BF_B) - tb(BF_\wedge))}{k \times \ln(1 - 1/m)} \end{aligned} \tag{3.2}$$

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 one. 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 optimal cluster, to show that moderate Bloom filter sizes are enough for high-quality clustering.

Theorem 3.2. *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. Without loss of generality, assume that $f(P_i, C_x) > f(P_i, C_y)$, where $f(P_i, C_j)$ denotes the clustering objective function for peer P_i and cluster C_j (Eqn. 3.1). Then, the probability of $|P_i \cap C_x| > |P_i \cap C_y|$ is at least:*

$$\begin{aligned} & Pr\left[|P_i \cap C_x| > |P_i \cap C_y|\right] > 1 - \\ & exp\left(-f(P_i, C_y) \times \left(\frac{2f(P_i, C_x) - 2f(P_i, C_y)}{2f(P_i, C_y) + f(P_i, C_x)}\right)^2 / 4\right) \times \\ & exp\left(-f(P_i, C_x) \times \left(\frac{f(P_i, C_x) - f(P_i, C_y)}{2f(P_i, C_y) + f(P_i, C_x)}\right)^2 / 2\right) \end{aligned} \tag{3.3}$$

The proof is included in the appendix.

An interesting observation from Theorem 3.2 is that correctness probability in-

creases 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.

3.4.2 Cluster centroid caching

Super peers are responsible for maintaining the Bloom filter for their cluster centroid. To avoid rebuilding these filters from scratch every time a peer joins or leaves a cluster, super peers maintain them using counting Bloom filters. 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. 3.4). 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 Bloom 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 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. 3.1), a cache peer only requires a counting filter of two bits per counter for finding both BF_{C_x} and $BF_{\{C_x - P_i\}}$. In other words, 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 counters, 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.

3.4.3 Aggregated Cluster Information Publishing

We enhance the distributed inverted index to enable efficient peer clustering. Apart from the peer-related information, each super peer additionally publishes aggregated cluster information. 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 the 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. 3.4).

Publishing of the cluster 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.

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 3.4: Logical DHT inverted index for clustering-enhanced PCIR. Changes compared to the basic PCIR index are grey-shaded.

3.4.4 Joining Peers in the Clustering-Enhanced Algorithm

As in the basic algorithm, peers participating in clustering-enhanced PCIR first join the DHT and then find a super peer to attach to. The two algorithms differ on 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 3.4.1). However, real-world 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. Note however that while a peer splits its content into several virtual peers, it participates always as a single node in the DHT, thereby not increasing the network size. This approach further reduces the average number of distinct terms per super peer, and therefore also reduces the overall maintenance cost for full-text indexing.

To cluster the documents and create the virtual peers we use standard K-Means. As a clustering objective function, K-Means uses Jaccard similarity [90], 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 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 related clusters (Alg. 3.3, lines 7-13). For each related cluster, the virtual peer computes the partial cosine similarity based on the retrieved scores. Then (lines 18-22), 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, computed using Equation 3.1. 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 14-16).

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 $\mu + \lambda \times \log(n)$ 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, 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

Algorithm 3.3: Peer periodically joining a peer cluster (clustering-enhanced PCIR)

```
   Input: myDocuments: Peer documents;
          numberOfVPs: Number of virtual peers per peer
   // break to virtual peers
1  virtualPeers ← KMeans(myDocuments,numberOfVPs);
2  for virtualPeer vp ∈ virtualPeers do
3      vp.BF ← ComputeMyBloomFilter(vp.localterms);
4      Map<Cluster,Score> clusterScores;
5      TopTerms < T_1, T_2, ... T_λ > ← TopSort(vp.localterms, λ);
6      for term t in TopTerms do
7          peer p ← DHTLookup(t.hashValue);
8          <Cluster,Score>relevantClusters ← p.getClusterList(t);
9          for <c:Cluster,s:Score> in relevantClusters do
10             if clusterScores.contains(c) then
11                 clusterScores(c) ← clusterScores(c)+s;
12             else
13                 clusterScores(c) ← s;
14     if clusterScores is empty then
           // become a super peer
15         vp.isSuperPeer ← true;
16         vp.clusterInvertedIndex ← myInvertedIndex;
17     else
18         Sort descending all clusters on their score;
19         for top-μ clusters c do
20             if c.canAcceptMe() then
                   // the cluster is not overloaded
21                 cp ← CachePeer(c) ;    // peer caching the centroid
                   // compare my bf to the cluster bf
22                 BFSimScore[c] ← cp.compare (vp.BF,cp.clusterBF);
23         Sort all clusters on the BFSimScore desc;
24         Join the cluster with the maximum BFSimScore;
```

same way as in the basic PCIR system, described in Section 3.3.

3.5 Cost Analysis

We now describe the cost model for the two 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 the Chord ring itself, as these are the same in all approaches.

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

3.5.1 Flat DHT Publishing

We first assess the cost for publishing all terms in a flat DHT setting. The expected cost in number of messages per DHT lookup 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:

$$C_{flat} = n \times D_p \left(\log(n) + 1\right) \tag{3.4}$$

3.5.2 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 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. 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 peers to find and join a group is $C_f = n\left(\log(n) + 1\right)$.

(b) Updating the DHT inverted index. Each super peer requires D_g DHT lookups, which cause a total of $D_g \times \log(n)$ 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 $C_u = n_{sp} \times D_g \left(\log(n) + 1\right)$ messages.

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

$$C_{basic} = C_f + C_u = n \times \log(n) + n + n_{sp} \times D_g \left(\log(n) + 1\right) \tag{3.5}$$

3.5.3 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 individually.

The total cost in the clustering-enhanced approach is the sum of: (a) C_f : the cost for the $\alpha \times n$ 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 $\lambda \times \log(n)$ 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 incurs 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 $C_f \leq \alpha \times n\,(\lambda \times \log(n) + 2\mu + 2)$.

(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:

$$\begin{aligned} C_{cluster} &= C_f + C_u \\ &\leq n \times \alpha\,(\lambda \times \log(n) + 2\mu + 2) + n_{sp} \times D_g\,(\log(n) + 1) \end{aligned} \tag{3.6}$$

3.5.4 Cost comparison

We now compare the expected cost of PCIR and the flat DHT approach. For this we assume, as is common in IR [20, 71], that the dictionary size of each peer follows Heap's law [71]. 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 $D_p \approx k \times len(DC(p))^{\beta}$, where k and β are parameters dependent of the language and text type.

Theorem 3.3. *Given a P2P network of n peers structured over a DHT, with dictionary size following Heap's law, with parameters k and β. 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 3.3 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 it follows that $E\left(\frac{D_g}{D_p}\right) = (n/n_{sp})^{\beta}$ (see derivation of Eqn. A.4, in the proof of Theorem 3.3). To get $load_{SP}$ expected terms per super peer, we set $D_g = load_{SP}$. This gives us $load_{SP} = D_p\left(n/n_{sp}\right)^{\beta}$. By solving the equation for n_{sp}, we find the number of required super peers such that their average load is $load_{SP}$: $n_{sp} = n\left(D_p/load_{SP}\right)^{\frac{1}{\beta}}$.

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 3.6 shows that sampling of a very small number of peers (0.3% of the total peers) is sufficient for estimating these values and 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 $\approx n_{sp} \times D_g \times \log(n)$. Each super peer routes on average $n_{sp} \times D_g \times \log(n)/n$ 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 $2D_g + n_{sp} \times D_g \times \log(n)/n + n/n_{sp}$.

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

$$\frac{C_{cluster}}{C_{flat}} \approx \frac{n_{sp} \times D_g\,(\log(n)+1)}{n \times D_p\,(\log(n)+1)} \tag{3.7}$$

Equation 3.7 cannot be simplified further, as we did for the ratio of C_{basic}/C_{flat}. The analysis for the basic PCIR approach 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.

3.6 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.

3.6.1 Experimental setup and evaluation criteria

The efficiency of the two PCIR algorithms was experimentally evaluated using real-world document collections. 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, the peers and their contents remained 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 imposed the following constraints at the PCIR peers: (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 constraints, churn has 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 [141], also employed in other popular P2P systems, e.g., ALVIS [103] (b) sk-STAT [117], which uses peer granularity indexing with additional meta-data, (c) mk-STAT [117], 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 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) [93] and the MEDLINE collection [112]. The results were very similar for the two datasets, so we present only the details for RCV1. RCV1 is a publicly available standard dataset in IR consisting of 802,253 newswire articles preprocessed using stemming and stopword filtering. 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 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 RCV1 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 cost of basic and clustering-enhanced PCIR with the cost 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:

- *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.

In the following sections we report average results over 6 repetitions. Each execution was let to run for 4 iterations (4 DHT periodic submissions), until the cost per iteration stabilized. Section 3.6.2 focuses on the experiments with peer granularity indexing, and Section 3.6.3 presents the results corresponding to the three additional publishing strategies.

3.6.2 Results for peer granularity indexing

Figures 3.5(a) and (b) plot the number of messages and transfer volume required by each approach for different network sizes. The X axis corresponds to the λ value and Y axis shows the network resources required by each approach, as a percentage of the cost of flat DHT submission. Notice that the flat DHT submission cost is constant, since it is independent of λ. Basic PCIR cost is also not directly relevant to λ, but it is related indirectly via the number of groups. As explained earlier, to get a fair comparison between the two PCIR variants, the number of groups at basic PCIR was equal to the number of clusters at clustering-enhanced PCIR, which depends on λ.

For clarity we include only results for 1000, 3000, and 5000 peers. The results are for an upper bound of 40000 terms per super peer, which is however never reached in these experiments. Hence, 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 experiment.

Our first observation is that network savings grow with network size, for both basic and clustering-enhanced PCIR. For example, for 1000 peers and $\lambda = 1$, the network

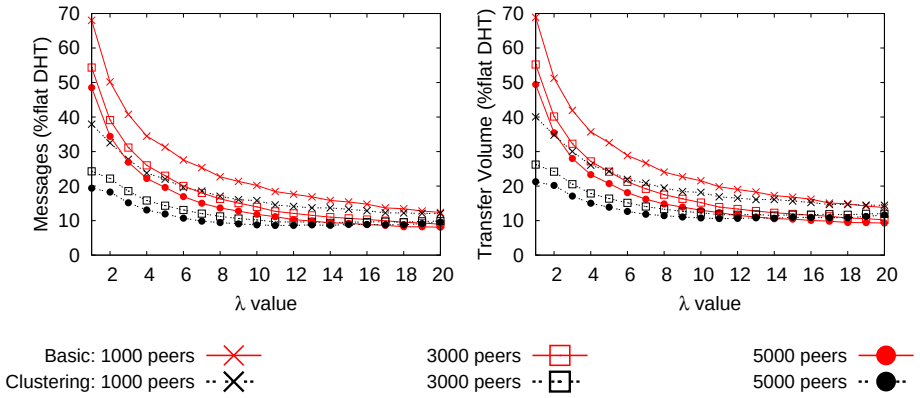

Figure 3.5: Number of messages and transfer volume (percentage of flat DHT) for PCIR with upper bound=40000 terms per super peer.

cost of basic and clustering-enhanced PCIR compared to the flat DHT approach is 70% and 40% respectively, whereas the ratios for 5000 peers are around 50% and 20%.

For $\lambda \leq 10$, clustering-enhanced PCIR is significantly better than basic PCIR. Its optimal performance is reached at $\lambda \approx 8$. Very low λ values ($\lambda \leq 3$) essentially limit the effectiveness of peer clustering, resulting in the creation of some small peer clusters. 90% of the optimal performance of clustering-enhanced PCIR is already obtained with $\lambda = 5$, whereas for $6 \leq \lambda \leq 10$, its performance remains 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 slightly increases because of the increase in the cost of peers to find and join their super peers.

Finding the optimal λ and μ values. As described in Section 3.5, the performance of PCIR, and thereby the λ and μ values for which clustering-enhanced PCIR minimizes the network cost, depend on the network size, the upper bound, and the collection. An online optimization similar to [174] 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. In 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 cost and the cost of PCIR with $\lambda = 6$ and $\mu = 2$ also decreases 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. Therefore, the rest of the results presented in this section are for $\lambda = 6$ and $\mu = 2$.

Varying the maximum terms per super peer. In practice, each super peer sets its upper bound itself, based on its network and computational capacity. In our experiments we assume that all super peers have the same network resources and set their upper bound to the same value.

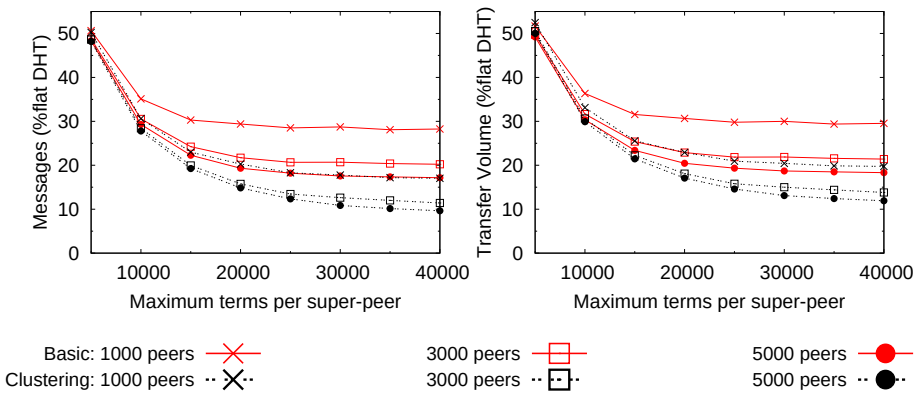

Figure 3.6: Number of messages and transfer volume (percentage of flat DHT) for PCIR: $\lambda = 6, \mu = 2$ and varying upper bound.

Figures 3.6(a) and (b) present the total maintenance cost (both super peers and standard peers) for different upper bound values, with $\lambda = 6$ and $\mu = 2$. For an upper bound of 5000 terms, clustering-enhanced PCIR has the same performance as 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 3.2 includes sample 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 shows that it is infrequent that the super peers acquire a dictionary larger than 20000 terms, and it happens for two reasons. First, the dictionary size per super peer follows the Heap's law (cf. Section 3.5), and therefore it grows slowly with the number of peers and documents in the cluster. Second, the employed objective function used during peer clustering assigns the virtual peers to

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

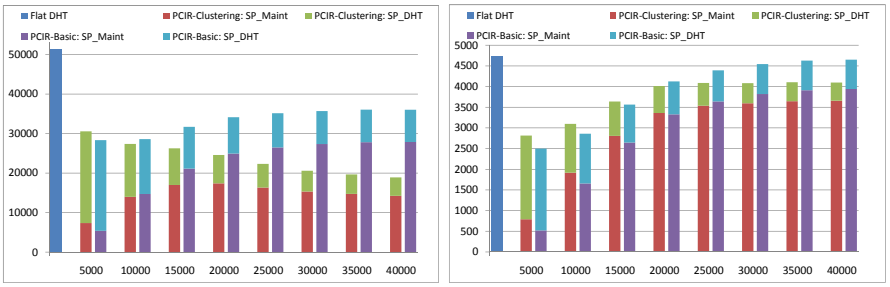

Figure 3.7: (a) Messages, (b) Transfer volume (Kbytes), for PCIR super peers, with $\lambda = 6, \mu = 2$, and for varying upper bound.

super peers such that the term overlap is increased. Therefore, each peer is expected to contribute 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. It is also important to ensure that super peers do not constitute a bottleneck for PCIR. We therefore analyzed the network workload of super peers in PCIR and compared it to the network load of regular peers, as well as to the load of the peers participating in flat DHT indexing. For clustering-enhanced PCIR, we measure the total workload of the physical peer which hosts the super peer and two 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. To get a clearer insight on the workload distribution, for both PCIR variants we distinguish between two types of network load:

***SP*$_{Maint}$:** Workload that is generated because of the role of the peer as super peer. This includes the cost of maintaining the peer groups/clusters, and of publishing the peer scores for all group/cluster terms.

***SP*$_{DHT}$:** Workload accounted to the participation of the super peer in the DHT, i.e., for routing DHT lookups which are issued from other peers.

Figures 3.7 (a) and (b) display 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). The X axis shows the upper bound of terms per super peer and the Y axis corresponds to the average network workload per super peer. For reference, the figure also includes the load per physical peer in the flat DHT approach which maintains the same DHT index.

We see that the average network workload of super peers of both PCIR variants is less than the respective workload of regular peers in the flat DHT indexing approach. Concerning number of messages, super peers at both PCIR variants have substantially less workload compared to regular peers in flat DHT indexing. With respect to transfer volume, super peers of basic PCIR have comparable cost with the physical peers at flat DHT indexing, whereas super peers of clustering-enhanced PCIR still have substantially less transfer volume compared to physical peers at flat DHT indexing.

Interestingly, 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. 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 additional messages for its super peer, it still needs to send its inverted index to its super peer, and this increases the transfer volume of the super peer. However, as explained in Section 3.3, each super peer can choose the upper bound for its workload independently so that its total workload does not exceed its capabilities.

Also notice that 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 3.2, which show that by increasing the upper bound to more than 30000 terms, only 5 clusters are affected. According to clustering-enhanced PCIR, when peers are not sufficiently similar to be clustered together, new clusters are created, so that centroids of existing clusters do not shift. Therefore, only super peers that have a centroid which is very similar to many peer centroids will notice an increase in their workload by increasing the upper bound.

As already noted, to keep the results independent of peer collection updates, peers did not use delta updating in our 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 indexes at their super peers by 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 3.5. 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 RCV1 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 3.3. The cost ratio computed experimentally was very close to the expected cost ratio. In particular, the maximum difference between the expected and

the experimentally derived value was 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. Nonetheless, even for the largest network of 5000 peers, sampling was still inexpensive as it involved only 15 peers.

Summarizing, the experimental results demonstrate that both PCIR approaches are significantly more efficient than flat DHT indexing. For large network sizes, the 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.

3.6.3 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 indexes, for increased retrieval quality [141]. Variants of document granularity indexing are used in well-known P2P systems, e.g., ALVIS [103], and therefore it is important to show that PCIR can support these systems. The other two approaches are sk-STAT and mk-STAT [117]. Both approaches employ peer granularity indexes, similar to Minerva [16]. To address the difficulty of collecting document granularity statistics to improve the IR quality, sk-STAT enhances the DHT postings by including all document identifiers corresponding to each term (e.g., their file names), represented as hash sketches. mk-STAT, which is built on top of sk-STAT, additionally employs query logs to identify and index in the DHT interesting term combinations, which are used for answering multi-term queries efficiently.

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 described in the corresponding original papers, and, (b) where the PCIR layer is additionally 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 [117]. The PCIR layer was configured with $\lambda = 6$ and $\mu = 2$.

Figure 3.8 presents the transfer volume for the three publishing strategies, for different upper bounds for the super peers. The presented results are for the largest network with 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. We see that PCIR offers substantial performance improvements with respect to the three alternative publishing strategies. In fact, the improvement is comparable to the peer granularity strategy, discussed in Section 3.6.2. The clustering-enhanced PCIR variant is again significantly better than its basic counterpart. Furthermore, similar to the results for the peer granularity strategy, PCIR performance increases with the super peers upper bound, and upper bounds higher than 30000 terms yield an order of magnitude lower cost compared to the baselines.

The publishing strategy does not influence the number of messages. Hence, the improvement concerning number of messages required by the three alternative pub-

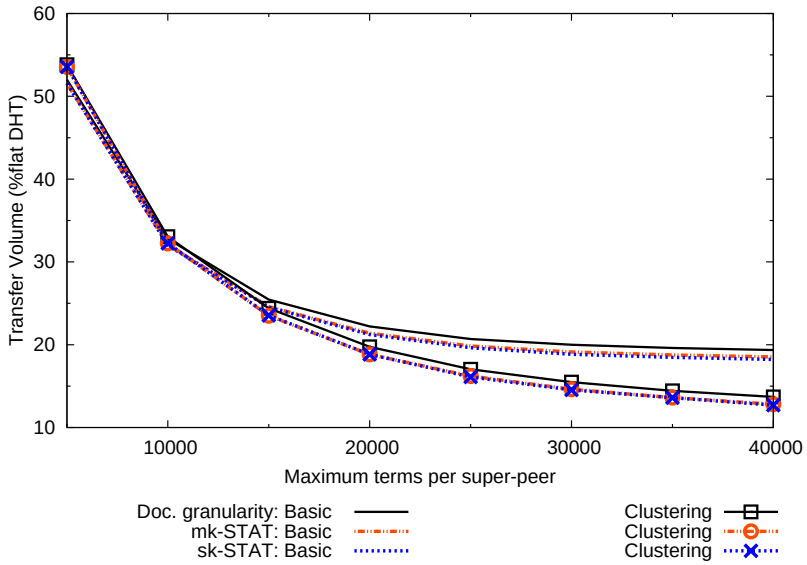

Figure 3.8: Transfer volume for different publishing strategies with PCIR

lishing strategies is equal to the peer granularity strategy, as reported in Section 3.6.2. This also holds for mk-STAT, even though it maintains additional multi-term keys in the DHT, because the additional information is attached to existing single-term maintenance messages.

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

3.7 Summary

In this chapter we have presented PCIR, a hybrid DHT/super peer system that significantly reduces the 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 aggregating all information about their group's collections and publishing it to the DHT. The performance improvement is achieved by exploiting the term overlap at the super peers to substantially reduce expensive DHT lookups.

The two variants of the algorithm differ in the method used for constructing the peer groups: the basic algorithm groups peers randomly around super peers, 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 a clustering algorithm for grouping, a significant term overlap within each group is observed, allowing for network savings of one order of magnitude. Compared to the basic algorithm, clustering-enhanced

PCIR further increases the term overlap in each group, and thus requires even less DHT lookups than the basic approach. Both approaches reduce the DHT maintenance network cost without having a negative influence on query execution, regarding both precision/recall, as well as performance of the original DHT model.

Experiments with real-world datasets demonstrate performance improvements of an order of magnitude over the conventional flat DHT approach, for four different publishing strategies. A theoretical analysis explains the benefits of our approach and supports our experimental results. Furthermore, super peers in PCIR 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.

Chapter 4

Distributed Indexing for Near Duplicate Detection

Efficiently and effectively searching for similar files over large file repositories is an active research topic, and was extended for file types beyond text, e.g., video [187], audio [180], and images [78]. This problem is generally known as *Near Duplicate Detection (NDD)*. Algorithms for NDD have various applications in large file repositories such as reduction of storage requirements [188], and detection of copyrighted multimedia content [177, 171].

Near duplicates are frequently created in P2P networks during normal file sharing operations, an example being different recordings of the same movie that may exist concurrently in a P2P file sharing network (for example, see Fig. 4.1). Minor differences between recordings may occur due to advertisements, lossy compression, different resolutions, or different recording programs. NDD can be used to find multiple sources/peers for downloading the same resource in parallel, to find the same video or audio resource at higher resolution, or to filter out near duplicates from the query results in order to present only novel results to a peer. P2P networks that focus on Multimedia Information Retrieval, such as SAPIR (www.sapir.eu) and VICTORY (www.victory-eu.org), also benefit from identifying near duplicates for enabling content-based retrieval of videos and audio files [51]. However, NDD methods implemented in these systems cannot scale to large P2P networks, since they assume a central repository, and are based on costly, pair-wise comparisons to detect the near duplicates.

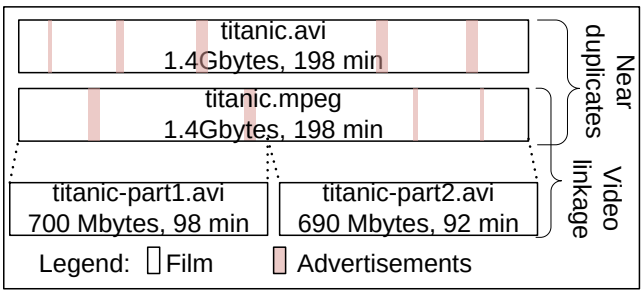

Figure 4.1: Near duplicates and video linkage.

In addition to centralized NDD systems, P2P and distributed NDD systems were also recently proposed, e.g., [14, 68, 51, 180]. Most of these employ a family of algorithms called *Locality Sensitive Hashing (LSH)* [39, 65] to map resources to bit strings, and build an index suitable for efficiently answering *K-Nearest Neighbor (KNN)* queries. NDD queries can be reduced to incremental KNN queries, e.g., keep querying for nearest neighbors until the difference threshold for near duplicates is surpassed. Although these approaches achieve good results in terms of effectiveness, they can be substantially improved in terms of efficiency by considering network and collection characteristics. Improving the performance of these approaches is the main objective of the work presented in this chapter.

In sharp contrast to previous approaches, we address NDD directly instead of considering it as a special case of KNN. This allows us to optimally choose the tuning parameters of LSH which affect the network cost and the probabilistic guarantees for the detection of the near duplicates. By tuning these parameters, we minimize the network cost for LSH, and we enable querying with constant cost and with probabilistic guarantees, both of which are not possible with existing P2P algorithms. Our approach, called *POND*, short for Peer-to-peer Optimized Near Duplicate detection, is both efficient and effective, and applicable to large P2P networks and to a variety of file types. In our experiments, the optimization of POND results in a cost reduction of several orders of magnitude.

In addition, we extend POND to cover a particular requirement of modern file sharing P2P networks: to link together all videos that have large near duplicate segments (e.g., "titanic.mpeg", with "titanic-part1.avi" and "titanic-part2.avi" in Fig. 4.1), a problem known as *Video Linkage (VL)* [79]. This video splitting occurs frequently, e.g., when burning large videos to CDs. By linking together these videos, we can parallelize further the downloading of large video files, and enable recovery of the downloading process even if all sources of the original video file are disconnected.

The rest of the chapter is organized as follows: We discuss existing work for NDD in Section 4.1. In Section 4.2 we present the preliminaries for near duplicate detection. We describe the P2P infrastructure used by POND in Section 4.3. Our main contribution, the adaptation of POND to the network and collection characteristics for minimizing the network usage under probabilistic guarantees, is presented in Section 4.4. Section 4.5 presents a large-scale experimental evaluation of POND, using a real-world collection of over 200 Gbytes, consisting of videos, audio, and text. We conclude in Section 4.6.

4.1 Related Work

Locality Sensitive Hashing (LSH) [65] has recently received substantial attention in the context of the K-Nearest Neighbor problem, as well as for near duplicate detection. The objective of LSH is to map similar elements to the same hash value with high probability, and dissimilar elements to different hash values, again with high probability (using distance thresholds for both "high" and "low" similarity). Bawa et al. [14] propose LSH Forest, which allows the distance thresholds to be set per query, and show how it can be deployed in a P2P network. However, LSH Forest does not optimize network usage for NDD queries. Since its objective is to enable the thresholds to be set per query, it cannot tune certain parameters of LSH (more specifically, number of hash tables l and number of hash functions k described in Section 4.2) which, as we will show later, are crucial for minimizing the total network cost. In fact, the cost of indexing

each resource in LSH Forest is $O(l \times k)$, which is even higher than the corresponding cost of standard LSH [65]. Furthermore, for query execution, the number of peers that need to be visited is not constant as in the standard LSH, but varies depending on the query parameters. In this work we have a different focus than LSH Forest. For the applications we consider, the distance thresholds do not change often; the thresholds may still change at run-time, but not necessarily per query. Having fixed thresholds enables us to select the values for k and l that minimize the network cost, and to save a substantial amount of network resources.

Similar to LSH Forest, several other P2P approaches address the NDD problem by employing a KNN infrastructure. In a recent work [68], Haghani et al. presented a P2P system based on LSH, and in particular on p-stable hashing. Their approach significantly reduces the cost compared to LSH Forest, as it requires only l indexes per resource. However, similar to LSH Forest, their work also focuses on KNN queries. As such, even though it can execute near duplicate queries by reducing them to incremental KNN queries, it does not optimize the LSH configuration for minimizing the network cost. Due to this lack of optimization, the cost for answering a NDD query is not constant. By restricting the problem domain to NDD queries only, the analysis and optimization presented in our work could also be applied to their system for optimizing the usage of network resources.

Dong et al. [44] focus on deriving analytically the optimal configuration for LSH with the objective of minimizing the computational time and maximizing the recall. They show significant performance increase compared to an unoptimized LSH. Unfortunately their approach only considers centralized scenarios for which the expensive resource is the computation time. As such, they do not consider the network usage as a resource, which is important for high-churn P2P networks like the ones we are interested in. Therefore, their results are not applicable to P2P.

There also exist methods on NDD and KNN for P2P systems that are not based on LSH. Falchi et al. [51] present DINN, an algorithm for incremental nearest neighbor in P2P based on priority queues, which however requires an efficient P2P range query, or an incremental P2P NDD implementation. Otherwise, DINN needs to route each query to almost all peers, and then cannot scale. Yang [180] proposes a P2P version of MACSIS, an algorithm for detecting near duplicate audio files. The algorithm is built over unstructured P2P networks, and uses gossiping for query execution; therefore, it cannot scale to more than a few hundred peers.

To the best of our knowledge, this work is the first to propose a scalable P2P algorithm that adapts to the peer contents, the network size, and the desired probabilistic guarantees for addressing the NDD problem with near-optimal network cost, without overloading the participating peers.

4.2 Prerequisites

In this section we provide an overview of near duplicate detection. To this end, we first formalize the objectives of near duplicate detection and video linkage. We then discuss representations for different multimedia resources, and finally describe Locality Sensitive Hashing (LSH) which forms the technical basis of PONDD.

4.2.1 Problem Definition

Each resource $x \in \mathcal{X}$ has a *type*, e.g., audio, video, text. Resources of the same type can have different formats, e.g., an audio resource can be mp3, wav, etc. A *resource representation* $R(x)$ is a normalized format-independent representation of resource x. In this work, we represent resources as sets of strings. The transformation which produces $R(x)$ from x depends on the specific resource type. We will summarize different suitable representations and transformations for video, audio, and text files in Section 4.2.2.

Definition 4.1 (Similarity function)**.** *Given two resources x_1 and x_2 of the same type $T \in \{audio, video, text\}$, the* similarity function *$Sim(R(x_1), R(x_2))$ computes the similarity of the two resources based on their resource representations. The similarity values are in the range* $[0, 1]$.

A similarity value of 0 means that the resources are completely dissimilar, while a similarity value of 1 denotes identical resources. In this work we measure similarity using Jaccard similarity, which is the standard similarity function for sets: $Sim(R(x_1), R(x_2)) := \frac{|R(x_1) \cap R(x_2)|}{|R(x_1) \cup R(x_2)|}$.

Definition 4.2 (Near duplicate resources)**.** *Given resources x_1 and x_2 of type T. The resources are* near duplicates *under similarity threshold t if they have $Sim(R(x_1), R(x_2)) \geq t$.*

Threshold t is in the range of $[0 \ldots 1]$. Its value depends on the application scenario. For instance, typical values for text are between 0.8 and 0.95 [173]. For our application scenarios, e.g., parallelizing downloads, or locating multimedia resources of different resolutions, this threshold is a fairly stable system property for each resource type. For example, downloading a video in parallel from two sources works only if the two sources have a similarity higher than a high threshold, otherwise merging of the two sources will produce unwanted artifacts in the video. The threshold can be determined a priori, according to the merging algorithm and the acceptable quality loss.

As explained already, it is often the case that videos have large overlapping segments, e.g., large videos may be partitioned into two or more files for easy storage. The smaller parts may additionally undergo post-processing, like compression or re-encoding with a different encoder. We want to be able to detect this relation between the partitioned files and the original file, and link all resources together.

Definition 4.3 (Video Linkage)**.** *Given video resources x_1 and x_2, the videos are* linked *if there exist non-empty segments of x_1 and x_2, denoted as $Segment(x_1)$ and $Segment(x_2)$, such that $Segment(x_1)$ and $Segment(x_2)$ are near duplicates.*

In this work we use 'part' to denote the physical partitioning of a video into two or more files (e.g.,titanic-part1.avi and titanic-part2.avi in Figure 4.1). In contrast, term 'segment' refers to a conceptual splitting of a video into smaller segments.

4.2.2 Resource Representations for NDD

POND can operate on different resource representations and similarity functions for each resource type. To keep the algorithm's description independent of the resource

types, we now introduce a common resource representation for all resource types, and describe appropriate transformations to convert each resource to this common representation. We also discuss about alternative representations and similarity functions, which may be preferred for some scenarios.

Text resources. We process a text document x to produce its representation $R(x)$ as follows: (a) extract all terms from x, (b) stem all extracted terms, and (c) filter out stopwords. Representation $R(x)$ consists of the set of remaining terms. This combination of the bag-of-words representation and Jaccard similarity is frequently used in IR for computing similarity of text documents.

Audio resources. The standard approach for evaluating similarity of two audio files consists of creating textual acoustic fingerprints of the two files, and then using text-based similarity measures to compute the similarity. Similar files are expected to have similar fingerprints. The most frequently used non-proprietary fingerprinting technique is *fooid* [89], integrated in several media players. Fooid is fast and portable, and produces fingerprints which are independent of the file format/encoding. Particularly, it produces fingerprints for 424 different dimensions, such as energy, tonality, and length. We convert these fingerprints to strings by concatenating the dimension id with the dimension value for each dimension. This results in a set of 424 strings for each audio file, which we use as a representation.

Video resources. In this work we use a representation of videos based on the color histograms of the keyframes [184]. The keyframes of a video are the frames that differ significantly from their preceding frames, i.e., more than a given threshold. To generate the video representation, we first detect the video keyframes, and compute their color histograms in the *HSV* space. We then generate a fingerprint of each histogram by finding and sorting the k most populated buckets, and concatenating their indices in a string. The video representation consists of the set of fingerprints of all keyframe histogram signatures. Video representations based on keyframe histograms are a standard technique for partitioning and indexing video resources [184], as well as for finding near duplicate videos [172, 177].

In order to address the video linkage problem, videos are conceptually split into segments. The splitting points are decided using the keyframe extraction algorithm described earlier, but using a higher distance threshold. Each video segment is handled as an individual resource, with its own representation. A record of the video segmentations is maintained by the peer that owns the video, such that the originating video of each segment can be found during query execution. Therefore, detecting one near duplicate segment for the query is sufficient for detecting the originating video, and for linking it with the query.

Alternative Resource Representations. The POND framework is not bound to the described similarity metrics and representations. For text files, it is straightforward to adapt POND to use the vector space model and cosine similarity by using the theoretical results of [27]. Regarding audio, other acoustic fingerprints can be used such as *fdmf* (`http://w140.com/audio/`), another open-source fingerprinting algorithm for which we observed similar results in our experiments. For video files, additional fingerprinting techniques based on keyframes [172, 177] are directly applicable to our framework.

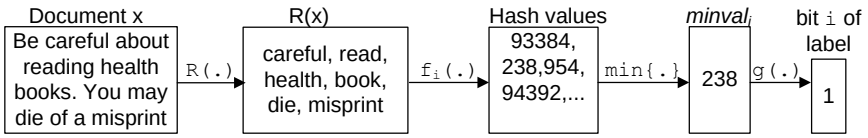

Figure 4.2: Computing the i bit of a label j for a text resource

4.2.3 Locality Sensitive Hashing

Locality Sensitive Hashing is often used for NDD in centralized environments. POND is also based on LSH which we briefly describe now. Our description follows the notation of [65].

LSH is based on the notion of locality sensitive hash families. Let $\mathcal{M}$ be a metric space (e.g., with dimensions corresponding to terms for text documents), and $d(\cdot,\cdot)$ the distance function defined for any two points of $\mathcal{M}$, (e.g., the Jaccard distance between two documents). A family of hash functions $\mathcal{H}$ is *locality sensitive* if there exist positive thresholds r_1 and r_2, and probabilities pr_1 and pr_2 for which the following conditions hold:

- If the distance between two points $d(p,q)$ (e.g., two text documents) is less than the distance threshold r_1, then $f(p) = f(q)$ with probability at least pr_1, where $f(\cdot)$ is randomly chosen from $\mathcal{H}$.
- If the distance between two points $d(p,q)$ is at least equal to r_2, then $f(p) = f(q)$ with probability at most pr_2, where $f(\cdot)$ is randomly chosen from $\mathcal{H}$.

We now demonstrate LSH using text documents as an example. The algorithm is initialized by constructing l hash tables $ht_1, ht_2, \ldots, ht_l$. To each hash table it binds k hash functions $f_1(\cdot)$, $f_2(\cdot)$, ... $f_k(\cdot)$, which are selected uniformly at random from $\mathcal{H}$. For inserting a document d in this structure, the document's representation $R(d)$ is used to compute l different labels of length k: $\mathcal{L}(d) = \{Label_1(d), Label_2(d), \ldots, Label_l(d)\}$. The algorithm computes $Label_j(d)$ that corresponds to hash table ht_j, in a bit-by-bit approach. Bit i of label $Label_j(d)$ is generated as follows (cf. Fig 4.2):

1. Hash function $f_i(\cdot)$ $(1 \leq i \leq k)$ corresponding to ht_j is used to hash each term from $R(d)$.
2. The minimum hash value $minval_i$ produced by $f_i(\cdot)$ for all terms is detected.
3. $minval_i$ is mapped to a binary value by further hashing. The resulting bit is used as the ith bit of the document's label.

For generating all k bits for $Label_j(d)$, this process is repeated for $i = 1, 2, \ldots, k$ resulting to the list $\mathcal{L}(d)$ of labels for d. Document d is then hashed in all l hash tables using the corresponding labels as keys.

For successful deployment of the LSH algorithm in P2P environments we require: (a) an inexpensive method to store the l hash tables in a P2P network, (b) a way to coordinate the peers so that they maintain the required hash tables, and, (c) a distributed and dynamic approach to adapt the configuration of LSH (parameters k and l) to minimize network costs. We will elaborate on these issues in the subsequent sections.

4.3 POND Infrastructure

The backbone of POND is an inverted index combining a Distributed Hash Table and Locality Sensitive Hashing. POND indexes the files/resources in this inverted index such that similar resources are indexed using the same DHT keys, so that they can be efficiently located using standard DHT lookups. The algorithm consists of two parts: (a) the *indexing part* (Section 4.3.1) which is responsible for configuring and maintaining the distributed index, and, (b) the *query execution part* (Section 4.3.2), responsible for efficiently locating the near duplicates of a query file.

The novelty of POND is the adaptation of the indexing process to the network and data collection characteristics, such that the required probabilistic guarantees are satisfied with the minimal cost. In this section, we only sketch this process. We will describe the configuration/optimization process in detail in Section 4.4.

4.3.1 Inverted Index Maintenance

In Section 4.2.3 we have explained how resource labels are computed for different file types. We now elaborate on how the distributed inverted index is configured and maintained, using the resource labels as keys. Our implementation relies on Chord for constructing an inverted index, described in Section 2.4. However, other DHTs could be used as well, since POND relies only on the basic DHT functionality, common to all DHT implementations.

Before initializing the index, the user requirements need to be determined: (a) the desired probabilistic guarantees pr_{min}, expressing the minimum probability that each near duplicate resource will be detected, and, (b) the minimum similarity *minSim* between two resources for considering them as near duplicates. POND then automatically determines the optimal system configuration, and constructs the index.

The index maintenance algorithm is divided in three steps: (1) algorithm configuration/optimization, where the core parameters are determined for optimizing the algorithm, (2) computing the labels for each resource, and, (3) indexing the resources in the DHT to enable their retrieval with subsequent near duplicate detection queries. These steps are repeated at regular intervals so that the LSH configuration remains optimal for the network size and for the content of the peers.

Step 1: Algorithm configuration/optimization The purpose of this step is to estimate the parameters of LSH that satisfy the probabilistic guarantees with minimum network cost. A randomly chosen peer p_i collects statistics from the network and computes the parameters that minimize the cost of POND: the optimal number l of labels per resource, and the length k of each label. Then, p_i uses the DHT overlay to broadcast k and l to all participating peers. This step is a key step for POND, and it will be described in more detail in Section 4.4.

Step 2: Computing the labels After receiving the updated l and k values, the participating peers compute the labels for their resources – l labels of k bits for each resource. Computation of labels is an inexpensive, local process. To optimize this step further, labels are cached and reused, even if they are computed for different k and l values. So, if either k or l is increased, only the difference between the existing labels and the new labels needs to be computed.

Step 3: Indexing After each peer has computed the labels for its resources, it needs to index each resource in the DHT-based inverted index. For each resource x and for

each label $Label_i(x)$, the peer publishes a triple $< i, IP, RecID >$, using $Label_i(x)$ as a key, where i is a label index, IP is the IP address of the peer, and $RecID$ is a resource ID for resource x. It is important to include i in this triple, because otherwise the peer holding the triple cannot distinguish whether a resource x and a query q have the same ith label, or whether $Label_i(x) = Label_j(q)$ with $i \neq j$. Clearly, the latter case does not imply anything about the similarity of x and q.

To handle *churn*, peers use periodic republishing. When a peer publishes a triple in the DHT, it attaches an expiration time. If the triple is not updated within its expiration time, it is automatically removed from the DHT. The expiration time of a triple comes shortly after the next expected republishing time; thereby resources are always indexed in the DHT, and obsolete resources are removed soon after they expire.

Note that the presented infrastructure is not the only possible infrastructure for POND. The optimization techniques described later, which is the main contribution of this work, can be applied with some modifications to other systems as well, e.g., [68, 14].

4.3.2 Querying for near duplicate resources

A peer p_q detects the near duplicates for a query resource q as follows. First it uses $R(q)$, the representation of q, to compute the l labels of q, denoted as $\mathcal{L}(q)$. For each label $Label_i(q) \in \mathcal{L}(q)$, the peer retrieves from the DHT inverted index the triples of all resources published using the same label as a key. These resources are the candidate near duplicates. Following, p_q sends $R(q)$ to all peers that hold each of the discovered resources, which then respond with links of all their near duplicate resources. In the response, only the resources that satisfy the minimum similarity with the query representation are included. As we will show in Section 4.4, this process guarantees that each near duplicate resource will be detected with probability greater or equal to pr_{min}.

The same algorithm is used for detecting video linkages. Recall from Section 4.2.2 that for video linkage, peers conceptually break each video to segments, and index each segment individually. With respect to query execution, the query video is broken into segments as well, and all near duplicates are retrieved for each segment. We will show in Section 4.5 that this approach is very effective. In practice, not all segments need to be queried; a peer stops querying as soon as it finds sufficient linkages for the application requirements, e.g., for efficiently parallelizing the download.

When the objective of the query is for finding more peers owning a file, for parallelizing its download, clearly the query initiator does not yet have the full file, and hence it cannot compute the resource representations. Therefore, it first downloads the resource representation from another peer that owns the full resource. We assume that peers already have an approach to find at least one copy of the file, for instance, using keyword search over a DHT index, as enabled by LimeWire and other mainstream applications. Since resource representations are very compact, even for large videos, the additional cost imposed by this process is negligible.

4.4 Configuration and Optimization of POND

Having described the overall framework of POND in Section 4.3, we now elaborate on the first indexing step, i.e., configuring and optimizing the POND parameters. We provide an overview of the step in Section 4.4.1. In Sections 4.4.2 and 4.4.3 we present the

n	Number of peers
$avgRes$	Average number of resources per peer
$minSim$	Minimum similarity for considering two resources as near duplicates
pr_{min}	Minimum probability that POND will detect each near duplicate resource
l	Number of labels per resource
k	Length of each label in bits

Table 4.1: Notations

theoretical analysis (network cost analysis and probabilistic analysis) which drives the network optimization of POND. Finally, in Section 4.4.4 we combine the cost analysis and probabilistic analysis to derive the optimal parameters l and k for LSH that minimize the total network cost and satisfy the desired probabilistic guarantees. Table 4.1 introduces the notations used in the following sections.

4.4.1 Overview on the Parameter Optimization Procedure

In this step, the algorithm determines the values for l and k that minimize the total network cost, and satisfy the desired probabilistic guarantees. The algorithm is executed at regular intervals, on randomly selected peers. The problems that need to be addressed are: (a) random selection of a peer for executing the optimization algorithm, with minimal coordination between the peers, (b) statistics gathering, (c) computation of the optimal values of l and k and their dissemination in all participating peers.

Random Selection. For the random selection problem, we require that one peer is randomly selected for executing the algorithm every m minutes, and that all peers have the same probability of being selected. We address this problem using a simple randomized algorithm which is resistant to churn and requires no coordination. More specifically, every m minutes, each peer decides with probability $1/n$ to execute the optimization, where n denotes the overall number of peers. Thus, the expected number of algorithm executions per m minutes is one.

Statistics Gathering. After a peer is selected to execute the optimization, it estimates the following statistics:

- the number of peers in network, using the approach proposed in [62]
- the number of resources per peer, and the query rate
- the probability distribution function (PDF) for all pairwise resource similarities in the corpus

The latter two are estimated using random sampling on a small percentage of peers, 1% in our experiments. Sampling requires a small number of messages and negligible transfer volume, thereby it does not overload the participating peers. This simple method already provides good results, as verified in our experimental evaluation, but alternative methods for estimating these statistics based on gossiping or random walks are equally applicable, e.g., [62].

Computation and dissemination of the optimal l, k. Using the described statistics, the peer computes the optimal values for l and k, as we will describe in detail in Section 4.4.4. The optimal configuration is then disseminated in the network, such that it can be further used for index maintenance and query execution. For disseminating the configuration efficiently, the peer constructs a message including the collected statistics and the estimated optimal values for k and l. The message is tagged with the local peer time, and broadcasted over the DHT using an approach proposed by El-Ansary et al. [48]. The algorithm requires $O(n)$ messages and $O(\log(n))$ time. Notice that it might happen that two peers are selected to perform parameter optimization almost simultaneously. This is not a problem for POND since during configuration dissemination only the most recent configuration is kept.

4.4.2 Cost analysis

The network cost of POND is composed as follows: (a) the cost of maintaining the DHT overlay, (b) the cost of maintaining the locality-sensitive inverted index over DHT, and (c) the cost of querying for near duplicates. The cost of establishing the links between peers to maintain the DHT overlay is orthogonal to POND, therefore we do not integrate it in our analysis. The reader can find a detailed cost analysis for Chord in [158].

Inverted index maintenance cost. Indexing one resource in the DHT requires at most $l(\log(n) + 1)$ messages:

- Executing l DHT lookups (one DHT lookup for each of the resource labels) requires at most $l \times \log(n)$ messages.
- Using the l labels as keys, and publishing the resource's meta-data in the DHT requires l additional messages.

Thus, the total number of messages for indexing all resources in the network is: $C_{maint} \leq n \times avgRes(l \times \log(n) + l) = O(n \times avgRes \times l \times \log(n))$.

Query execution cost. Let q denote the resource for which we wish to find the near duplicates, $ND(q)$ the set of all near duplicates for q, and $FP(q)$ the set of resources that are falsely identified as near duplicates (the false positives). Finding the near duplicates of q requires $C_{find} \leq l(\log(n) + 2) + 2|ND(q)| + |FP(q)|$ messages:

- Executing a DHT lookup for each of the l labels of q requires a maximum of $l \times \log(n)$ messages.
- Retrieving the candidate resources for the l labels requires $2l$ additional messages.
- Sending $R(q)$, the representation of the query, to the peers holding all candidate near duplicates requires a maximum of $|ND(q)| + |FP(q)|$ messages.
- Retrieving all near duplicate resources requires a maximum of $|ND(q)|$ messages.

The total cost per republishing period is a linear combination of C_{maint} and C_{find}. Let y denote the expected number of queries at each republishing period. Then, the total cost is $C_{total} = C_{maint} + y \times C_{find}$.

We now have the cost expressions for all aspects of the algorithm. In the following sections we show: (a) how to get an estimate for $|FP(q)|$, and, (b) how to choose the two parameters of POND, l and k, such that the total cost is minimized.

4.4.3 Probabilistic Analysis

We first compute the probability that a near duplicate resource will be detected. Then, we derive an expectation for the number of false positives per query. These two parts are put together in Section 4.4.4 to determine the combination of k and l that minimizes the cost.

The following lemma computes the probability that two resources x and y have the same label $Label_j$, corresponding to hash table ht_j.

Lemma 4.1. *Given resources x and y with similarity $Sim(x, y)$. The probability that the jth label of x is identical to the jth label of y is:*

$$Pr[Label_j(x) = Label_j(y)] = \sum_{i=0}^{k} \binom{k}{i} \times (1 - Sim\,(x, y))^i \times Sim(x, y)^{k-i} \times 0.5^i$$

The proof is included in the appendix.

Note that the probability $Pr[Label_j(x) = Label_j(y)]$ is the same for all $j \in [1, \ldots, l]$. For convenience we will denote this probability as $Pr\,[Label(x) = Label(y)]$.

The probability that two resources x and y have at least one common label is given by the following corollary.

Corollary 4.1. *Given resources x and y with similarity $Sim(x, y)$. The probability that x and y have at least one common corresponding label is given by*

$$\begin{aligned} pr_{\text{found}}(x, y) &= 1 - (1 - Pr\,[Label(x) = Label(y)])^l \\ &= 1 - \left(1 - \sum_{i=0}^{k} \binom{k}{i} \times (1 - Sim\,(x, y))^i \times Sim(x, y)^{k-i} \times 0.5^i\right)^l \end{aligned} \tag{4.1}$$

The proof is included in the appendix.

Finally, the probability that two near duplicate resources x and y have at least one common label can be lower bounded as follows.

Theorem 4.1. *Given resources x and y with similarity $Sim(x, y) \geq minSim$. The probability that x and y have at least one common corresponding label is at least equal to:*

$$pr_{\text{ndd}}(x, y) \geq 1 - \left(1 - \sum_{i=0}^{k} \binom{k}{i} \times (1 - minSim)^i \times minSim^{k-i} \times 0.5^i\right)^l \tag{4.2}$$

The proof is included in the appendix.

POND will detect two resources as potential near duplicates if these have at least one common corresponding label. Therefore, the probability that each near duplicate will be detected by the algorithm corresponds to the probability given by Equation 4.2.

Estimating the number of false positives. The false positives for a query q are the resources detected by the algorithm as potential near duplicates but have similarity with q lower than *minSim*. Peers detect that these resources are false positives before transferring them over the network. Nevertheless, as explained earlier, false positives cause additional network overhead, for detecting and filtering them out. Therefore, we need to estimate the number of false positives for the purpose of accurately modeling and minimizing the network cost.

We use $\mathcal{S}$ to denote the set of all resources in the network having similarity with q less than *minSim*. Using Corollary 4.1, the expected number of false positives $E(|FP(q)|)$ can be computed as follows:

$$E(|FP(q)|) = \sum_{x \in \mathcal{S}} \left(1 - (1 - Pr\left[Label(x) = Label(q)\right])^l\right)$$

The estimation of the number of false positives using the above equation is costly to compute as it requires computing all similarities between the query and all available resources. An efficient alternative is to estimate $|FP(q)|$ using an approximation for the probability distribution of similarities. Let $p(x)$ denote the probability distribution function *(PDF)* of the pairwise similarities of all resources in the corpus. Then, the expected number of false positives is:

$$E(|FP(q)|) = avgRes \times n \times \int_0^{minSim} p(x) \times pr_{labels}(x)\ dx$$

where $pr_{labels}(x)$ denotes the probability that two resources with similarity x share at least one label, and is computed using Equation 4.1. For a discrete PDF, we derive an analogous expression by replacing the integral with a sum.

The PDF $p(x)$ depends on the corpus and on the chosen similarity measures. Our experiments with large collections (Section 4.5) indicate that the Zipf distribution constitutes a good fit when using Jaccard similarity and the representations described earlier. For estimating the Zipfian coefficient, POND uses peer sampling during the optimization step, as described earlier in this Section.

4.4.4 Minimizing the Network Cost of POND

We are now ready to find the values for k and l that minimize the network cost. Recall from Section 4.3 that POND should find each near duplicate with a probability at least pr_{min}. By reducing Equation 4.2 for k and solving for $pr_{\text{ndd}} = pr_{min}$ we find the value of k that will return each near duplicate resource with a probability higher than pr_{min}. This value, denoted as k_0, is:

$$k_0 = \frac{\log\left(1 - (1 - pr_{min})^{1/l}\right)}{\log\left(0.5 - \frac{1}{2\ minSim}\right) + \log(minSim)}$$

Next, we show that k_0 is also the value for k that minimizes the network cost.

Theorem 4.2. *For given l, minSim and pr_{min} the value of k that minimizes network cost is $k = \lfloor k_0 \rfloor$*

The proof is included in the appendix.

So far, we assumed that the optimal value for parameter l is given. We now show how to eliminate this assumption. Finding theoretically the value of l that minimizes the cost is complicated. Therefore, we use the following algorithm to compute the optimal l. We start with $l = 1$, and find the optimal k according to Theorem 4.2. For these pairs of k and l we compute the expected cost, according to the analysis presented in Section 4.4.2. We then increment l, and repeat the computations. For some value of l, denoted with l_{inc}, the expected cost will start to increase, i.e., the cost for $l = l_{inc}$ will be higher than the cost for $l = l_{inc} - 1$. The optimal value of l is $l_{inc} - 1$. This holds because the cost function is convex, and therefore has only one minimum, namely at $l = l_{inc} - 1$. Note that this algorithm is executed once per republishing period, at the peer responsible for the optimization. It is computationally inexpensive and requires no network resources; thus, it does not constitute a bottleneck.

4.5 Experimental Evaluation

The purpose of our experimental evaluation was threefold. First, to investigate the significance of the dynamic optimization performed by POND, by comparing it with an NDD algorithm which does not optimize the values of k and l. Second, to examine the efficiency and effectiveness of POND using real-world collections and system configurations. Third, to evaluate the proposed extension for video linkage.

4.5.1 Datasets and Evaluation Setup

POND was evaluated on three large, real-world collections. As a text collection we have chosen the standard Reuters Corpus Volume I (RCV1) [93], one of the standard IR datasets, including more than 800,000 newswire articles. For videos, we constructed a collection by crawling a sample of Youtube videos with Tubekit [150].[1] The collection contains 22,455 videos with a total volume of 144 Gbytes. We did not consider existing publicly available video benchmarks, such as TRECVID, because these contain a relative small number of videos (less than 200), and therefore cannot be used for simulating P2P networks of reasonable sizes. For audio, we extracted the audio tracks of the described Youtube collection as MP3s, amounting to a total size of 82 Gbytes. The size of the three datasets summed up to 227 Gbytes.

For the text collection, we simulated a P2P network of 100,000 peers, structured over Chord. All articles were distributed uniformly among the participating peers. For the video and audio collections which had less resources, we simulated a network of 1000 peers, distributing the resources uniformly among peers as well.

For constructing the query set, we compared all resources pair-wise to identify the pairs with similarity higher than a threshold *minSim*, for $minSim \in \{0.8, 0.9\}$. Some of these pairs were in fact *exact* duplicates, e.g., the same video with a different file name. POND detected all exact duplicates, since these were always producing identical labels. Since these resources could also be detected with traditional hashing techniques, they were excluded from the query set for our experiments.

For each configuration, we measured the total network cost, i.e., the number of messages required for both maintenance and query execution. Effectiveness was computed using the standard *recall measure*, i.e., the percentage of detected near duplicates. There was no need to measure precision separately, since this is always 1, due to the

[1] We would like to thank the author of Tubekit, Chirag Shah, for assisting us with the crawling task.

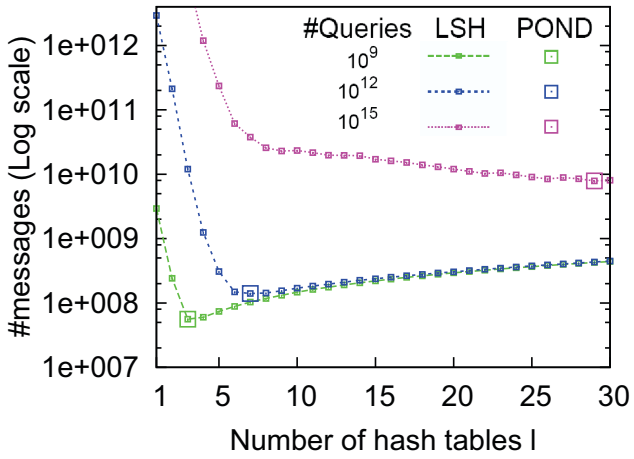

Figure 4.3: Cost versus number of hash tables l, for $minSim = 0.9$

way POND collects the near duplicates. As explained in Section 4.3, resources which are not near duplicates are efficiently filtered out and are neither transmitted over the network to the query initiator, nor presented to the user as a result of the query.

4.5.2 Comparison with LSH

The advantage of POND compared to previous LSH algorithms is that it dynamically optimizes the values of k and l so that the total network cost is minimized. To find out the importance of this optimization for P2P networks, POND was compared with a P2P implementation of standard LSH, in which k and l are chosen manually by the user. In particular, POND was compared with `LSH(K)`, described in [14]. Since the procedure of choosing l and k is the only difference between `LSH(K)` and POND, the difference observed in the efficiency of the two algorithms can be directly attributed to the optimization performed by POND.

To this end, we have indexed the RCV1 collection in a network of 100,000 peers, and varied the rate of queries per republishing period between 10^6 and 10^{15}. Since `LSH(K)` does not offer an approach for choosing l and k, it was tested for all possible combinations of l and k satisfying $pr_{min} = 0.9$. For each configuration we measured the total network cost, i.e., the aggregate cost for maintenance and query execution. The observed costs are shown in Figure 4.3 (note that the Y axis is in logarithmic scale). For clarity, the figure includes only the most efficient `LSH(K)` configuration for each l value.[2] The cost incurred from POND for the same pr_{min} is a single point, since POND selects exactly one k and l combination for each query rate.

We observe that the value of l strongly influences the efficiency of LSH. On the one hand, setting l too low leads to a very small k, increasing the number of false positives by several orders of magnitude. On the other hand, setting l too high imposes the unnecessary overhead of maintaining more hash tables. The difference between the optimal and the incurred cost for each setting can be several orders of magnitude. More importantly, we see that there is no universally optimal value for l and k, but the

[2]For a given pair of l and k values that satisfy the probabilistic guarantees, all other configurations with the same l and with $k' < k$ also satisfy the probabilistic guarantees, albeit with higher network cost.

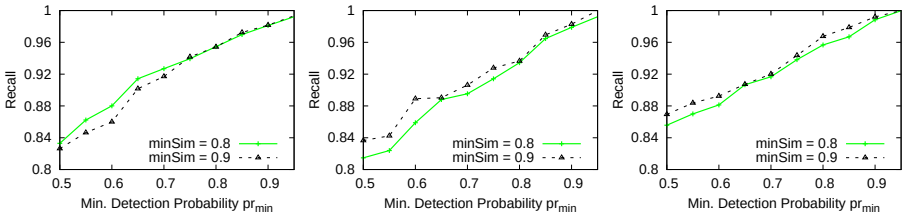

Figure 4.4: Recall versus minimum detection probability pr_{min}: (a) RCV1 collection, (b) Video collection, (c) Audio collection

optimal values depend on the system properties, which significantly vary with time in P2P networks [146]. In contrast, the default cost of POND for each setting is always nearly equal to the global minimum cost of the original LSH; the extra network cost induced by the POND optimization step is negligible. This means that POND always finds the best configuration for l and k, and minimizes the cost.

The experiment was repeated with the two multimedia collections, as well as with different network sizes. The experimental results were similar to the presented ones, verifying the importance of optimizing the values of l and k. In conclusion, by optimizing the values of l and k for the given configuration, POND can reduce the total network cost by several orders of magnitude.

4.5.3 Effectiveness for near duplicate detection

Effectiveness for NDD is measured with recall (precision is always 1, as explained in Section 4.5.1). Figure 4.4 presents recall for different values of pr_{min}, and for $minSim \in \{0.8, 0.9\}$. As expected, increasing pr_{min} results to an increase on recall. In particular, for higher pr_{min} values, the optimization step constructs more hash tables or reduces the hash functions in order to satisfy the higher probability requirements. This behavior is consistent for the three different resource types, as well as for different *minSim* values.

We also see that even for very low pr_{min} values, recall remains within acceptable levels. For example, recall is always higher than 0.8, even for $pr_{min} = 0.5$. The reason for this is that POND computes the probabilistic guarantees considering the minimum similarity value *minSim*, whereas most of the near duplicates have similarity higher than *minSim*. Therefore, the actual probability for finding most of the near duplicates is higher than the obtained probabilistic guarantees, yielding a recall higher than the required one.

4.5.4 Network cost

Figure 4.5 plots the network cost of POND for different values of pr_{min}, with *minSim* set to 0.9. We distinguish between two types of cost: (a) *maintenance cost*, which is the average number of messages required for indexing a single resource in the network, and, (b) *query cost*, corresponding to the average number of messages required for detecting and retrieving all near duplicates of a query, and filtering out all non near-duplicates.

We observe that both indexing cost and query execution cost remain within reasonable limits. For example, for the RCV1 experiment with $pr_{min} = 0.8$, maintenance requires less than 40 messages per article, whereas querying requires less than 80. Note

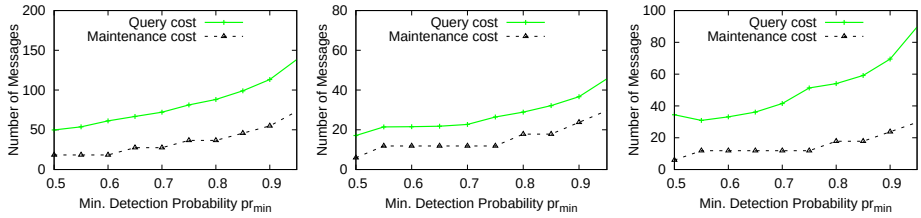

Figure 4.5: Cost versus minimum detection probability pr_{min} (a) RCV1 collection, (b) Video collection, (c) Audio collection

that for the aforementioned configuration, recall is already above 0.95. Furthermore, as expected, increasing pr_{min} causes higher network cost, because POND constructs more hash tables to satisfy the required probabilistic guarantees. Nevertheless, even for $pr_{min} = 0.95$ which provides a recall of almost 1, maintenance cost does not surpass the 70 messages per resource.

Also note that query execution cost is always higher than maintenance cost. As described in Section 4.4.2, maintenance cost involves performing l DHT lookups, while query execution cost additionally requires contacting all candidate peers and collecting all the near duplicates. Therefore, maintenance cost per resource is expected to always be less than query execution cost.

It is also interesting to compare the costs incurred for different types of resources. We observe that the cost related to text documents is slightly higher than the respective cost for video and audio resources. This difference is mostly attributed to the difference of network sizes in the corresponding experimental setups, as this affects the cost of DHT lookups. Since POND uses DHT lookups both for maintenance and query execution, the respective costs are also affected. Nevertheless, the increase is only logarithmic to the network size. We also observe that the querying costs for audios and videos differ slightly, even though the audio resources are generated from the videos. This happens because there are more near duplicates in the audio collection, compared to the video collection. The typical case in which two audio files were near duplicates while their corresponding videos were not, was songs accompanied by slide-shows, i.e., two videos having the exact same song as audio but showing completely different slide-shows.

4.5.5 Video Linkage

POND was also evaluated with respect to video linkage. For generating the query set Q, we have selected a subset of videos with file sizes uniformly distributed between 20 and 120 Mbytes, and divided them to 2, 3, or 4 equal-sized parts. We then used all partial videos $q \in Q$ as queries for POND. A query was considered successful if it had returned the original video from which it was created.

Figure 4.6 plots the average recall for different video file sizes. POND was initialized with $pr_{min} = 0.9$ and $minSim = 0.9$, on a network of 1000 peers. We observe that recall increases significantly with the size of the original video. For example, recall for the smaller files (20 – 40 Mbytes) is around 0.75, whereas recall for larger files (100 – 120 Mbytes) is nearly 1. This is due to the video segmentation method. Since each query segment is handled as an individual resource, the probability that a query succeeds increases exponentially with the number of overlapping segments between

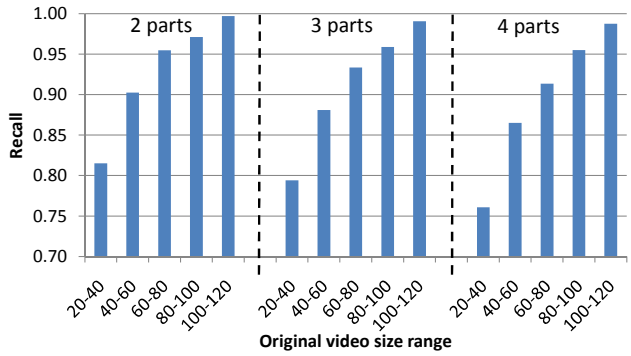

Figure 4.6: Video linkage: Recall versus average file size for videos broken to 2, 3 and 4 parts

the query and the full video. For extremely small queries, the overlapping segments are too few, if any, yielding a lower recall. For the same reason, recall decreases with the number of divisions for creating the query. By increasing the video and query size, the number of common segments increases, and the recall reaches practically 1 for the videos above 100 Mbytes. Note that these are the videos for which finding the near duplicates would be more important, e.g., for parallel downloading in file sharing networks.

With respect to network cost, the maintenance cost for video linkage is typically higher than the cost of NDD because each video segment is indexed and looked-up individually. In our experiments, maintenance cost for video linkage was at most 110 messages per video, which is easily affordable. This cost is distributed uniformly among all participating peers, as it is mostly composed of DHT lookups. Additional cost optimizations can be achieved by reducing the number of segments in each video, i.e., by increasing the distance threshold for splitting a video to segments. Including this threshold in the optimization analysis is part of our future work.

In conclusion, POND efficiently and effectively addresses the video linkage problem, and focuses especially on large videos which constitute the dominant majority in the Internet, achieving a recall very close to one.

4.6 Summary

In this chapter we have presented POND, a novel algorithm for near duplicate detection whose key innovation lies in the deployment of parameter optimization to minimize network usage. We have derived probabilistic guarantees for the success of POND to answer NDD queries, and showed how the algorithm automatically configures the core parameters of the underlying indexing method. Furthermore, we extended POND to address the video linkage problem. A large-scale experimental evaluation using real world datasets of more than 200 Gbytes demonstrated the importance of optimizing the LSH parameters for reducing the cost of NDD in P2P networks. The results show that POND successfully optimizes the LSH parameters, reduces the network cost to the theoretical minimum, and satisfies the required probabilistic guarantees. The incurred network cost is easily affordable by the participating peers, even for huge networks and collections, making the algorithm suitable for integration in any mainstream DHT-based file-sharing system.

Chapter 5

P2P Text Clustering

In this chapter we propose an algorithm for P2P text clustering. Text clustering is widely employed for automatically structuring large document collections and for enabling cluster-based information browsing, to alleviate the problem of information overflow. It is especially useful in highly distributed environments such as distributed digital libraries [75] and P2P information management systems [2], since these environments operate on large-scale document collections, scattered over the network. Several P2P systems rely on text clustering to enhance information retrieval efficiency and effectiveness [101, 176, 83].

Most existing text clustering algorithms are designed for central execution. They require that clustering is performed on a dedicated node, and are not suitable for deployment over large scale distributed networks. Therefore, specialized algorithms for distributed and P2P clustering have been developed, such as [42, 47, 69, 73]. However, these approaches are either limited to a small number of nodes, or they focus on low dimensional data only. Hence, a distributed clustering algorithm that scales to large networks and large text collections is required.

We are particularly interested in systems where the content distribution is imposed by their nature, such as P2P desktop sharing systems [2, 119], distributed digital libraries [75, 70], and mainstream file sharing P2P networks. Clustering in such networks is challenging, firstly because the data is inherently distributed and no participant has the capacity, or willingness, to collect and process all data, and secondly, because of the high churn rate, affecting availability of content and of computational nodes. For these systems, we require a P2P algorithm that can cluster distributed and highly dynamic text collections, without overloading any of the participating peers, and without requiring central coordination.

A key factor to reduce network traffic in these systems is to reduce the number of required comparisons between documents and clusters. Our approach achieves this by applying probabilistic pruning: Instead of considering all clusters for comparison with each document, only a few most relevant ones are taken into account. We apply this core idea to K-Means, one of the frequently used text clustering algorithms. The proposed algorithm, called Probabilistic Clustering for P2P (PCP2P), reduces the number of required comparisons by an order of magnitude, with negligible influence on the clustering quality.

In the following section, we review the basic algorithms employed by PCP2P. Section 5.2 gives an overview of existing distributed clustering approaches. In Sections 5.3 and 5.4 we introduce PCP2P and present the corresponding cost analysis. We describe

Algorithm 5.1: K-Means

```
Input: allDocs: documents to be clustered;
       k: number of clusters
1 allClusters ← selectRandomClusters(k) ;
2 for Document d in allDocs do
3     for Cluster c in allClusters do
4         sim ← cosineSimilarity(d, c) ;
5     Assign(d, cluster with maximum Similarity) ;
```

the probabilistic analysis in Section 5.5, and show how PCP2P is parameterized to achieve a desired correctness probability. In Section 5.6 we verify experimentally the scalability and quality of PCP2P, with simulations of up to 1 million peers and 1 million documents, using real and synthetic data. We conclude the chapter in Section 5.7.

5.1 Prerequisites

In this section we briefly describe the infrastructure and algorithms used by PCP2P.

Document model. Similar to most clustering algorithms, PCP2P represents the documents and clusters using the Vector Space Model. For document d and term t, we denote the frequency of t in d with $TF(t,d)$. As usual, we apply term stemming and stopword filtering to reduce the document model sizes and allow for more meaningful clustering. Since Vector Space Models represents clusters and documents as high-dimensional vectors, all standard norms can be used for computing the vector length. The ones most interesting for data mining, which are frequently used for normalizing the term frequencies, are the L1-Norm and the L2-Norm. The L1-Norm of a cluster or document x is denoted as $|x|_1$ and is defined as the sum of all term frequencies in the cluster/document, $|x|_1 = \sum_{t \in x} TF(t,x)$. The L2-Norm is denoted as $|x|$ and defined as $|x| = \sqrt{\sum_{t \in x} TF^2(t,x)}$.

K-Means Clustering. K-Means [105], which we approximate with PCP2P, is one of the most frequently used clustering algorithms because of its low complexity (linear in the number of documents) and high clustering quality, particularly for text clustering. The basic K-Means algorithm (cf. Alg 5.1) can be summarized as follows: (1) Select k random starting points as initial centroids for the k clusters. (2) Assign each document to the cluster with the nearest centroid. (3) Recompute the centroid of each cluster as the mean of all cluster documents. (4) Repeat steps 2-3 until a stopping criterion is met, e.g., no documents change clusters anymore.

For document clustering, K-Means shows comparable or better performance than competitive algorithms while incurring significantly less cost [155].

DKMeans Clustering. A direct distribution of centralized clustering algorithms does not scale to large networks. We still sketch such an approach here, to point out where our optimizations take place. Like its centralized counterpart, distributed K-Means (*DKMeans*) requires maintaining the cluster centroids, and comparing all documents to all clusters to determine the best cluster. In DKMeans, the responsibility of maintaining

Algorithm 5.2: DKMeans

```
   Input: myDocs: Peer documents to be clustered
1  allClusters ← findAllClusters() ;
2  for Document d in myDocs do
3      for Cluster c in allClusters do
4          Send termVector(d) to ClusterHolder(c);
5          Sim[c] ← cosineSimilarity(d,c);
6      Assign(d, argmax_c Sim[c]) ;
```

each cluster centroid is assigned to a randomly selected peer, called *cluster holder*. As in K-Means, each cluster holder selects a random document as initial centroid. Clustering is performed as follows (cf. Algorithm 5.2): A peer first identifies all cluster holders, e.g., using a DHT index (Line 1). Then, for each of its documents, it sends its term vector to each cluster holder for comparison with the centroid and receives a similarity score (Lines 4, 5). It then assigns the document to the most similar cluster, by notifying the respective cluster holder (Line 6).

DKMeans requires that at each clustering iteration, all documents are sent over the network to all cluster holders, for comparison. This clearly prohibits the algorithm to scale. In Section 5.3 we show how PCP2P addresses this issue by drastically reducing the number of required comparisons.

5.2 Related Work

Several algorithms for parallelizing K-Means have been proposed, e.g., [43, 56], which harness the power of multiple nodes to speed up the clustering of large datasets. These however assume a controlled network or a shared memory architecture, and therefore are not applicable for P2P, where these assumptions do not apply.

Other approaches focus on P2P text clustering [40, 41, 42, 47, 69, 73, 83]. Eisenhardt et al. [47] proposed one of the first P2P clustering algorithms. The algorithm distributes K-Means computation by broadcasting the centroid information to all peers. Due to this need for broadcasting, the algorithm does not scale to large networks. Hsiao and King [73] avoid broadcasting by employing a DHT to index all clusters using manually selected terms. This approach requires extensive human interaction for selecting the terms, and the algorithm cannot adapt to new topics.

Hammouda et al. [69] use a hierarchical topology for the coordination of K-Means computation. Clustering starts at the lowest level of the hierarchy, and the local solutions are aggregated until the root peer is reached. This algorithm has the disadvantage that clustering quality decreases noticeably for each aggregation level, because of the random grouping of peers at each level. Therefore, quality decreases significantly for large networks. Already for a network of 65 nodes, the authors report a quality of less than 20% of the quality of K-Means.

The state of the art in P2P clustering is the proposal of Datta et al. [41, 42]. In particular, the authors proposed LSP2P and USP2P, two P2P approximations of K-Means. LSP2P uses gossiping to distribute the centroids. In an evaluation with 10-dimensional data, LSP2P achieved an average misclassification error of less than 3%. However, as we show in Section 5.6, when it comes to clustering text, LSP2P fails because it is based on the assumption that data is uniformly distributed among the peers, i.e., each

peer has a representative set of documents from each cluster. This assumption clearly does not hold for text collections in P2P networks. The second algorithm, USP2P, uses sampling to provide probabilistic guarantees. However, the guarantees are also based on the assumption that data is uniformly distributed among the peers. Furthermore, USP2P requires a coordinating peer which gets easily overloaded, since it is responsible for exchanging centroids with a significant number of peers, for sampling, e.g., 500 peers out of 5500 peers.

The problem of forming clusters of peers, as opposed to clusters of documents, is also considered in the literature. For example, in Chapter 3 of this thesis, we have proposed PCIR which employs peer clustering to speed up the maintenance of distributed inverted indexes. Koloniari and Pitoura [83] discussed P2P clustering in the context of optimizing information retrieval in flat P2P networks. However, these approaches do not focus on optimizing the network and computational cost of clustering per se, and would therefore be too expensive when applied in the context of document clustering. The algorithm proposed in this chapter is also directly applicable for peer clustering, assuming that the contents of each peer can be represented with the vector space model.

5.3 PCP2P: Probabilistic Clustering for P2P

We start with a high-level description of the algorithm, followed by a detailed explanation of each step.

In PCP2P, a peer undertakes up to three different roles. First, it serves as document holder, i.e., it keeps its own document collection, and it assigns its documents to clusters. Second, it participates in the underlying DHT by holding part of the distributed index, and by routing DHT lookup messages. Third, a peer may become a *cluster holder*, i.e., maintain the centroid and document assignments for one cluster.

PCP2P consists of two parallel activities, *cluster indexing* and *document assignment*. Cluster indexing is performed by the cluster holders. In regular intervals, these peers create compact cluster summaries and index them in the underlying DHT, using the most frequent cluster terms as keys. We describe this activity in Section 5.3.1. The second activity, document assignment, consists of two steps, preselection and full comparison. In the *preselection step*, the peer holding d retrieves selected cluster summaries from the DHT index, to identify the most relevant clusters (Alg. 5.3, Line 2). Preselection already filters out most of the clusters. In the *full comparison step*, the peer

Algorithm 5.3: PCP2P: Clustering the documents

```
   Input: myDocs: Peer documents
1  for Document d in myDocs do
       // PRESELECTION STEP
2      CandClusters ← CandidateClustersFromDHT() ;
       // FULL COMPARISON STEP
3      RemainingClusters ← FilterOut(CandClusters) ;
4      for Cluster c in RemainingClusters do
5          Send termVector(d) to ClusterHolder(c);
6          Sim[c] ← cosineSimilarity(d,c) ;
7      Assign(d, cluster with maximum similarity) ;
```

computes similarity score estimates for d using the retrieved cluster summaries. Clusters with low similarity estimates are filtered out (Line 3), and the document is sent to the few remaining cluster holders for full similarity computation (Lines 4-6). Finally, d is assigned to the cluster with highest similarity (Line 7). This two-stage filtering reduces drastically the number of full comparisons (usually less than five comparisons per document, independent of the number of clusters). Both cluster indexing and document assignment are repeated periodically to compensate churn, and to maintain an up-to-date clustering solution.

The algorithm enables controlling the tradeoff between the network cost and the clustering quality. In particular, the cluster indexing activity, as well as the preselection and full comparison steps, are configured using the results of a probabilistic analysis, thereby providing probabilistic guarantees that the resulting clustering solution exhibits nearly the same quality as centralized clustering.

In the next section, we further describe the process of indexing the cluster summaries. In Section 5.3.2 we look into the document assignment process, whereas in Section 5.3.3 we present the three filtering strategies in detail. Section 5.3.4 discusses further aspects of the algorithm.

5.3.1 Indexing of Cluster Summaries

Cluster holders are responsible for indexing the summaries of the clusters in the DHT. Particularly, each cluster holder periodically recomputes the centroid of the cluster, using the documents assigned to the cluster at the time. It also recomputes a *cluster summary* and publishes it to the DHT index, using selected cluster terms as keys. As we explain later, this enables peers to efficiently identify the relevant clusters for their documents. For this identification, it is sufficient to consider the most frequent terms of a cluster c as keys, i.e., all terms t with $TF(t,c) \geq CluTF_{min}(c)$, where $CluTF_{min}(c)$ denotes a frequency threshold for c. We use $TopTerms(c)$ to denote the set of these terms. Note that $TopTerms(c)$ does not include stopwords; these are already removed when building the document vectors. For illustration purposes, for the remainder of this section we assume that $CluTF_{min}(c)$ is known. In Section 5.5 we explain how the optimal value for this threshold is derived, such that the algorithm satisfies the desired probabilistic guarantees.

The cluster summary (cf. Fig. 5.1) includes: (1) all cluster terms in $TopTerms(c)$ and their corresponding TF values, (2) $CluTF_{min}(c)$, and, (3) the sum of all term frequencies (the L1-norm), cluster length (the L2-norm), and dictionary size. Note that we choose not to normalize the term frequencies with inverse document frequencies (IDF), because of the high cost associated for maintaining an IDF index over a P2P network, compared to the low influence in clustering quality [132]. Nevertheless, existing techniques for estimating IDF, such as [168, 9, 124], or techniques to circumvent

TopTerms				
Term	football	tennis	basketball	rugby
TF	21	19	17	17
Cluster Statistics				
L1=1603 **L2**=82.2 **DictSize**=120				
ClusterHolderIP=173.203.120.1 $CluTF_{min}$= 17				

Figure 5.1: Sample cluster summary

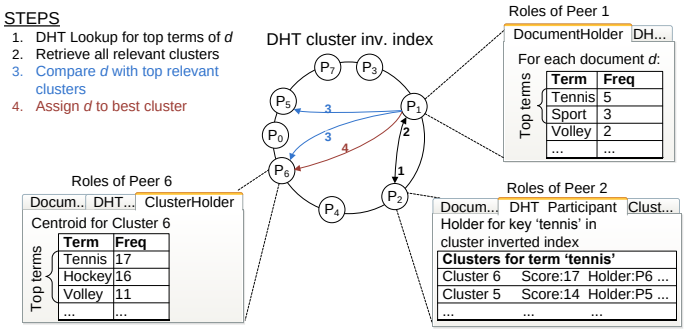

Figure 5.2: PCP2P System architecture

the lack of IDF, such as using the Inverse Peer Frequency [37] can also be combined with PCP2P.

To avoid overloading, each cluster holder selects random peers to serve as *helper cluster holders*, and replicates the cluster centroid to them. Their IP addresses are also included in the cluster summaries, so that peers can randomly choose a helper for comparing their documents with the cluster centroid without going through the cluster holder. Communication between the master and helper cluster holders only takes place for updating the centroids, by exchanging the respective local centroids, as in [43]; therefore, load balancing does not impede the system scalability.

5.3.2 Document Assignment to Clusters

Each peer is responsible for clustering of its documents periodically. Clustering of a document consists of two steps: (a) the preselection step, where the most promising clusters for the document are detected, and, (b) the full comparison step, where further clusters are filtered out, and the document is fully compared with the remaining clusters and assigned to the best one.

Preselection step. The preselection step works as follows. Consider a peer p, clustering a document d. Let $TopTerms(d)$ denote all terms in d with $TF(t,d) \geq DocTF_{min}(d)$, where $DocTF_{min}(d)$ denotes a frequency threshold for d (we explain how $DocTF_{min}$ is derived in Section 5.5). For each term t in $TopTerms(d)$, peer p performs a DHT lookup and finds the peer that holds the cluster summaries for t (Fig. 5.2, Step 1). It then contacts that peer directly to retrieve all summaries published using t as a key (Step 2). To avoid duplicate retrieval of summaries, p executes these requests sequentially, and includes in each request the cluster ids of all summaries already retrieved. Then, p merges the responses and constructs a list with all retrieved cluster summaries. We refer to this list as the *preselection list*, denoted with C_{pre}.

As we will show later, the summary of the optimal cluster for d is included in C_{pre} with high probability. This probability can be controlled by choosing the value of $DocTF_{min}$ for each document, and the value of $CluTF_{min}$ of each cluster. In Section 5.5 we explain how the participating peers decide on these values automatically, such that the desired probabilistic guarantees are satisfied.

Full comparison step. Peer p then sends the term vector of d to the candidate cluster holders for performing full document-cluster comparison, and retrieves the comparison results (Fig. 5.2, Step 3). The list of candidate clusters C_{pre} may be too long. To avoid sending the document to all cluster holders for full comparison, p uses the cluster summaries contained in C_{pre} to filter out the clusters not appropriate for the document at hand. In the following section we describe three different strategies for this task. Finally, p assigns d to the most similar cluster, and notifies the respective cluster holder (Fig. 5.2, Step 4).

5.3.3 Filtering Strategies

We propose three different strategies to filter out clusters from C_{pre}: (a) Conservative, (b) Zipf-based, and, (c) Poisson-based filtering. All strategies use the information contained in the retrieved cluster summaries to estimate the cosine similarity between a document and each of the clusters, and filter out the unpromising clusters. The conservative strategy makes a worst-case similarity estimation, which guarantees that the optimal cluster from the ones in C_{pre} will be detected. The Zipf-based and Poisson-based strategies filter out the clusters more aggressively, thereby reducing the network cost substantially, at the expense of a small probability of errors.

Conservative Filtering. This strategy works similar to Fagin's No Random Access (NRA) algorithm [50] for top-k selection: clusters whose maximum possible score is less than the best already known score are removed from the candidate clusters.

In our case the actual score is the cosine similarity between document and cluster centroid, defined as

$$Cos(d,c) = \sum_{t \in d} \frac{TF(t,d) \times TF(t,c)}{|d| \times |c|} \tag{5.1}$$

where $|d|$ and $|c|$ are the corresponding L2-Norms, and TF denotes the term frequency of the term in the document/cluster.

Since the actual value of $Cos(d,c)$ is not available, conservative filtering employs the information contained in the cluster summaries to estimate the cosine similarity between the document and each candidate cluster, denoted as $ECos(d,c)$. To ensure that the optimal cluster will not be filtered out, conservative filtering sets the value of $ECos(d,c)$ to the maximum possible value for the actual similarity $Cos(d,c)$.

The similarity estimate relies on term frequency estimates for all document terms not included in the cluster summary. It is computed according to the following rules: (a) each term not included in the summary of a cluster c has a term frequency of at most $CluTF_{min}(c) - 1$, otherwise it would have been included in the summary, and, (b) the sum of all estimated term frequencies in the cluster cannot exceed the actual cluster length. Precisely, let $t_1, \ldots, t_z$ denote the terms of d, sorted descending on their frequency, such that $TF(t_i,d) \geq TF(t_j,d)$ for all i, j s.t. $i < j$. For cluster c and term t, we denote the estimated term frequency as $ETF(t,c)$.

Then, cosine similarity is estimated as

$$ECos(d,c) = \sum_{x=1}^{z} \frac{TF(t_x,d) \times ETF(t_x,c)}{|d| \times |c|} \tag{5.2}$$

Particularly for the conservative strategy, $ETF(t_x, c)$ is computed as follows:

$$ETF(t_x, c) = \begin{cases} TF(t_x, c) & \text{, if } t_x \in TopTerms(c) \\ \min\left(CluTF_{min}(c) - 1, |c|_1 - ST(c) - SE(t_x, c)\right) & \text{, otherwise} \end{cases}$$

where $|c|_1$ denotes the L1-Norm of the cluster terms (see Section 5.1). We use $ST(c)$ to denote the sum of cluster frequencies for all terms included in $TopTerms(c)$, i.e., $ST(c) = \sum_{t \in TopTerms(c)} TF(t, c)$, and $SE(t_x, c) = \sum_{y=1}^{x-1} ETF(t_y, c)$ is the sum of all term frequencies for c estimated up to now.

The filtering process works as follows. Peer p sends the compressed document vector to the first cluster holder in C_{pre}, denoted as $c_{selected}$, and retrieves the actual cosine similarity $Cos(d, c_{selected})$. It then removes from C_{pre} the summary of $c_{selected}$, as well as the summaries of all clusters with estimated similarity $ECos(d, c) < Cos(d, c_{selected})$. The process is repeated until C_{pre} is empty, and the document is assigned to the cluster with the highest cosine similarity. Since the expected cosine similarity is never underestimated, conservative strategy always assigns the document to the optimal cluster.

For the process to complete faster, the list of clusters is sorted based on their lower bound for the cosine similarity. This bound is computed from the information included in the corresponding cluster summary, which is already retrieved by the peer. We call this the partial cosine similarity:

$$PCos(d, c) = \sum_{t \in TopTerms(c)} \frac{TF(t, d) \times TF(t, c)}{|d| \times |c|} \tag{5.3}$$

Using this ordering has a positive effect on the performance of the full comparison step.

Zipf-based filtering. Although conservative filtering substantially reduces the number of comparisons, it is based on a worst-case estimate for the cosine similarity. A more optimistic estimate allowing for further reductions can be derived based on the assumption that the term frequencies in the cluster follow a Zipf distribution. A term frequency estimate for the cosine similarity can be computed as follows:

$$ETF(t_x, c) = \begin{cases} TF(t_x, c) & \text{, if } t_x \in TopTerms(c) \\ \min\left(|c|_1 - ST(c) - SE(t_x, c), \frac{|c|_1}{r(t,c)^s \times H_{DT(c),s}}\right) & \text{, otherwise} \end{cases}$$

$DT(c)$ denotes the number of distinct terms in c, and $H_{DT(c),s}$ the generalized harmonic number of order $DT(c)$ of s, i.e., $\sum_{i=1}^{DT(c)} i^{-s}$. Assuming that term frequencies follow a Zipf distribution with exponent s, the expected term frequency of t in c is given by $|c|_1 / (r^s \times H_{DT,s})$, where $r(t, c)$ represents the estimated rank of t in c. The ranks of missing terms are estimated as follows: the ith document term not included in $TopTerms(c)$ is assumed to exist in the cluster, with rank $r = |TopTerms(c)| + i$.

Apart from the definition of $ETF(t, c)$, Zipf-based filtering is identical to conservative filtering.

Poisson-based filtering. This strategy follows a different approach for pruning the candidate clusters, which allows for probabilistic guarantees, and for a tradeoff between clustering quality and network cost. The strategy is based on the assumption that the score of each term t across all documents follows a Poisson distribution, where the

score of a term represents the term frequency normalized on the document's length, i.e., $TF(t)/|d|$. Poisson distribution is often used, e.g., [164], as it provides a reasonable fit for the term scores, and it has some convenient properties, which will be discussed in Section 5.5.5.

Note that the Poisson distribution model does not contradict the well accepted Zipf model for term frequencies within a single cluster, since the two distributions model two discrete concepts. On the one hand, Poisson distribution models the scores of any term t in all clusters. On the other hand, the Zipf distribution describes the scores of all terms in a single document. Therefore, the two models are not contradicting.

Consider peer p, which has already retrieved C_{pre} for document d. First, p computes the partial cosine similarity $PCos(d, c)$ of d and each cluster $c \in C_{pre}$ (Eqn. 5.3). For the cluster c_{maxp} with the best partial cosine similarity, it sends d to the cluster holder to compute the full cosine similarity $Cos(d, c_{\text{maxp}})$. Then, p estimates the remaining cosine similarity $RCos(d, c)$ for all other clusters $c \in C_{pre}$, i.e., the difference between the actual cosine similarity $Cos(d, c)$ and the partial cosine similarity $PCos(d, c)$. Instead of estimating $RCos(d, c)$ directly, which would be inefficient, the peer computes the Probability Density Function (PDF) describing $RCos(d, c)$, which allows computing the probability that $RCos(d, c)$ is above a threshold. In particular, as we show in Section 5.5.5, $RCos(d, c)$ follows a Poisson distribution with parameter $\lambda = \sum_{t \in d \setminus TopTerms(c)} TF(t, d) \times \frac{AvgSc(t, C_{pre})}{|d|}$, where $AvgSc(t, C_{pre})$ is:

$$AvgSc(t, C_{pre}) = \sum_{c \in C_{pre}} \begin{cases} TF(t, c)/|c| & \text{, if } t \in TopTerms(c) \\ \frac{CluTF_{min}(c)-1}{|c|} & \text{, otherwise} \end{cases}$$

Using this distribution, peer p can compute the probability that $RCos(d, c)$ exceeds any value $Z \in [0, 1]$. This probability is (Section 5.5.5):

$$Pr\left[RCos(d, c) \geq Z\right] \leq \sum_{i=\left\lfloor \frac{m(1-Z)}{MaxSc} \right\rfloor}^{m} \exp(-\lambda) \times \frac{\lambda^i}{i!}$$

with $MaxSc = \frac{CluTF_{min}(c)-1}{|c|}$.

Following, p discards all clusters $c \in C_{pre}$ for which the remaining cosine similarity is less than the difference $diff = Cos(d, c_{\text{maxp}}) - PCos(d, c)$, with high probability. In particular, for a required correctness probability Pr_{fcs}, clusters are discarded if $Pr[RCos(d, c) < diff\,] \geq Pr_{fcs}$. The optimal cluster for the document is therefore kept with a probability higher than Pr_{fcs}. Peer p sends d to the remaining candidate clusters for cosine similarity computation, and assigns the document to the cluster with the highest cosine similarity.

Compared to conservative filtering, Poisson-based filtering discards clusters more aggressively, and thus reduces the required cosine similarities substantially. Its main benefits compared to Zipf is that it allows controlling the tradeoff between cost and clustering quality, and provides probabilistic guarantees. Therefore, it can reduce network usage significantly, while still exhibiting a low and predictable error probability.

5.3.4 Further aspects

In this section we describe how PCP2P handles churn, and how it initializes.

Churn. As already stated, cluster indexing and document assignment are repeated periodically to compensate churn and to maintain an updated clustering solution. No synchronization is required between the peers. Cluster holders rebuild the cluster centroids in regular intervals, i.e., every m minutes, including only documents that were assigned to them during the last m minutes. Therefore, if a peer gets disconnected, its documents are automatically removed from the clusters in the next m minutes. Peers re-cluster their documents also every m minutes. Hence, each document belongs to exactly one cluster at any time, and cluster centroids are at most m minutes stale. An up-to-date clustering solution can be obtained at any time by fetching the centroids from the cluster holders.

Initialization. There are different possible ways of initialization. The easiest one assumes that a peer from the network starts the algorithm by selecting k peers randomly to act as cluster holders. These cluster holders choose one of their documents as initial centroid, and publish the cluster summary to the DHT. Then, the initiating peer uses broadcasting over DHT to notify all participating peers to start clustering. Another approach can be used for a continuously clustering network [101]. In this case, initialization in the strict sense is not required. When a peer detects that a cluster holder has become unreachable, it takes over that role. Since documents are clustered periodically, the new cluster will become populated with the right documents in the next iteration.

5.4 Cost Analysis

We now compute the network cost of PCP2P, expressed in number of messages and transfer volume. The network cost is composed of the following: (a) cost of indexing the cluster summaries, (b) cost of the preselection step for each document, and, (c) cost of the full comparison step.

Indexing of the cluster summary requires performing a DHT lookup for each term in $TopTerms(c)$, and publishing the summaries. The corresponding indexing cost per cluster is $Cost_{ind} = O(|TopTerms(c)| \times \log(n))$, both with respect to transfer volume and number of messages. The preselection step is executed for each document d, and causes $|TopTerms(d)|$ DHT lookups, which translate to a cost $Cost_{pre} = O(\log(n) \times |TopTerms(d)|)$. Finally, for the full comparison step, the document is sent to all remaining candidate clusters. Let C_{fcs} denote the set of these clusters for d. Then, the full comparison step requires $Cost_{fcs} = O(|C_{fcs}|)$ messages. With respect to transfer volume, this step incurs a cost $TVCost_{fcs} = O(|C_{fcs}| \times |d|)$.

The cost of indexing the cluster summaries is negligible, because the number of clusters is small, and their summaries are very compact. The dominating cost, both with respect to number of messages and transfer volume, is the one incurred for assigning documents to clusters, namely $Cost_{pre} + Cost_{fcs}$. Per document, this cost has the following properties: (a) it grows logarithmically with the number of peers, because DHT access cost grows logarithmically, and (b) it is independent of the size of the document collection. It also depends on $|C_{fcs}|$, which is by definition at most equal to the number of clusters. Therefore, the clustering cost per document scales, in the worst case, linearly with the number of clusters. However, our experimental evaluation shows that the number of clusters in C_{fcs} is very small, and independent of the total number of clusters k. Therefore, in practice, this cost grows at a much slower rate than linear, enabling PCP2P to scale to large networks, and to large numbers of documents and clusters.

5.5 Probabilistic Analysis

For Section 5.3, we have assumed that the optimal values for $CluTF_{min}(c)$ and $DocTF_{min}(d)$ are given. We now describe how each peer can compute these values dynamically to satisfy the desired system-wide clustering quality requirements.

The analysis uses a probabilistic document generation model [130, 157]. Briefly, the model assumes that each document belongs to a topic $\mathcal{T}$, and each topic $\mathcal{T}_i$ is described by a term probability distribution ϕ_i (a language model). Composite topics are also possible. A document of length l belonging to $\mathcal{T}_i$ is created by randomly selecting l terms with replacement from ϕ_i. The probability of selecting a term t is given by the topic distribution ϕ_i.

5.5.1 Notations and Overview

The user initiating the algorithm chooses the desired quality in terms of the required probability that each document is clustered at the optimal cluster, $Pr_{correct}$. The rest of the system parameters are determined automatically from PCP2P. $Pr_{correct}$ is associated with the following three system parameters: (a) Pr_{ind}, that represents the probability that the cluster summary of each cluster c_i is indexed in all $TopDistr(\alpha, \phi_i)$, (b) Pr_{pre}, expressing the probability that the preselection step for a document retrieves the optimal cluster for this document, and, (c) Pr_{fcs}, which is the probability that the full comparison step retrieves the optimal cluster. The desired values for these probabilities are common to all peers, and they are selected automatically such that $Pr_{pre} \times Pr_{fcs} \geq Pr_{correct}$.

The purpose of the probabilistic analysis is to enable PCP2P to automatically set Pr_{ind}, Pr_{pre}, and Pr_{fcs} for the given $Pr_{correct}$, and to configure each step accordingly such that these probabilities are satisfied. First, during the cluster indexing step, the cluster holder of each cluster c_i computes the value of $CluTF_{min}(c_i)$ so that the cluster summary is indexed in each term from $TopDistr(\alpha, \phi_i)$ with the predefined probability Pr_{ind}. In Section 5.5.3 we show how $CluTF_{min}$ is computed per cluster such that Pr_{ind} is satisfied. A document is clustered correctly when both the preselection and full comparison step include the optimal cluster in their result. The two steps succeed with probabilities Pr_{pre} and Pr_{fcs}, respectively. At the preselection step, the peer holding document d selects $DocTF_{min}(d)$ so that the optimal cluster for d is retrieved with probability Pr_{pre}, taking the probability Pr_{ind} into consideration. We explain how $DocTF_{min}$ is computed in Section 5.5.4. Finally, the full comparison step takes place. In this step, p decides which candidate clusters to filter out as non-promising such that the best cluster is found with probability Pr_{fcs}. We present the corresponding probabilistic analysis for this step in Section 5.5.5.

The following notations are used throughout the analysis. With $C_{sol} := \{c_1, \ldots, c_k\}$ we denote a snapshot of clusters on an ongoing clustering. Each cluster $c_i \in C_{sol}$ follows the language model ϕ_i. We use $t_1[\phi_i], \ldots, t_n[\phi_i]$ to denote the terms of ϕ_i sorted by descending probabilities. With $TopDistr(\alpha, \phi_i)$ we denote the set of α terms with highest probability in ϕ_i, i.e., $t_1[\phi_i]$, ..., $t_\alpha[\phi_i]$. Table 5.1 summarizes the notations.

5.5.2 Preliminaries

We first derive probabilistic bounds for the estimation of the term frequency of any term t in a document or a cluster, given the term frequency distribution of the document or the cluster. These bounds are required for the subsequent analysis.

k	Number of clusters
ϕ_i	Term distribution of cluster c_i
$TopDistr(\alpha, \phi_i)$	The set of α terms with highest probability in ϕ_i
$t_x[\phi_i]$	The x term of distribution ϕ_i, where terms are sorted by descending probability
$Pr_{correct}$	Desired correctness probability
Pr_{ind}	Probability that the cluster summary is indexed in all terms in $TopDistr(\alpha, \phi_i)$
Pr_{pre}	Probability that the preselection step succeeds
Pr_{fcs}	Probability that the full comparison step succeeds

Table 5.1: Notations

Theorem 5.1 computes the probability that a term with a given expected term frequency has an actual term frequency in the document higher than a given value.

Theorem 5.1. *Given a document d which follows language model ϕ_i. The expected term frequency of term t in d according to ϕ_i is denoted with $\hat{TF}(t,d)$. For term t with $\hat{TF}(t,d) > DocTF_{min}$, the probability of the actual term frequency $TF(t,d)$ exceeding $DocTF_{min}$ is at least $1 - \exp(-\hat{TF}(t,d) \times (1 - DocTF_{min}/\hat{TF}(t,d))^2/2)$. Furthermore, for a probability Pr_{min} the term frequency of term t in d is at least $\lfloor \hat{TF}(t,d) - \sqrt{2\hat{TF}(t,d) \times \log(\frac{1}{1-Pr_{min}})} \rfloor$.*

Proof is included in the appendix.

Similarly, Theorem 5.2 computes the probability of a term with a given expected term frequency to have a term frequency in the cluster exceeding a given value.

Theorem 5.2. *Given a cluster c_i which follows language model ϕ_i. The expected term frequency of term t in c_i according to ϕ_i is denoted with $\hat{TF}(t,c_i)$. For term t with $\hat{TF}(t,c_i) > CluTF_{min}$, the probability of the actual term frequency $TF(t,c_i)$ to exceed $CluTF_{min}$ is at least $1 - \exp(-\hat{TF}(t,c_i) \times (1 - CluTF_{min}/\hat{TF}(t,c_i))^2/2)$. Furthermore, for a probability Pr_{min} the term frequency of term t in c_i is at least $\lfloor \hat{TF}(t,c_i) - \sqrt{2\hat{TF}(t,c_i) \times \log(\frac{1}{1-Pr_{min}})} \rfloor$.*

Proof is included in the appendix.

5.5.3 Indexing of Cluster Summaries

We now show how cluster holders derive the value of $CluTF_{min}$ so that the cluster indexing process satisfies the required probabilistic guarantees. Assuming that term frequencies follow a Zipf distribution (validated, for example, in [20]) the expected frequency of the α th most probable term of ϕ_i in a given cluster c_i is $\hat{TF}(t_\alpha[\phi_i], c_i) \approx |c_i|_1/(\alpha^s \times H_{m,s})$ where $|c_i|_1$ is the L1-norm of c_i (see Section 5.1), m is the number of distinct terms in c_i, and s is the Zipf exponent of ϕ_i.

The cluster holders are required to publish their cluster summaries using each of the terms in $TopDistr(\alpha, \phi_i)$ as a key, with a probability at least equal to the system-wide value Pr_{ind}. Notice that the corresponding cluster distribution ϕ_i for each cluster c_i, hence also the set of terms $TopDistr(\alpha, \phi_i)$, are unknown to the cluster holder. However, the cluster holder can use Theorem 5.2 to compute a lower bound for the

cluster term frequency of $t_\alpha[\phi_i]$:

$$CluTF_{min}(t_\alpha[\phi_i], c_i) = \left\lfloor \hat{TF}(t_\alpha[\phi_i], c_i) - \sqrt{2\hat{TF}(t_\alpha[\phi_i], c_i) \times \log\left(\frac{1}{1 - Pr_{ind}}\right)} \right\rfloor \quad (5.4)$$

All terms $t_j[\phi_i]$ for $j \leq \alpha$ will have a term frequency in the cluster at least equal to $CluTF_{min}$ with a minimum probability Pr_{ind}. Thus, cluster holders will detect each term in $TopDistr(\alpha, \phi_i)$ with a minimum probability Pr_{ind}.

In Section 5.5.6, we describe how the system-wide values for Pr_{ind} and α are set, and how the Zipf exponent is estimated.

5.5.4 Preselection Step

A peer p needs to set the value of $DocTF_{min}$ per document which will guarantee that the optimal cluster for the document is detected in the preselection step with a probability Pr_{pre}. We now describe how this value is determined.

According to the cluster indexing activity, the cluster holder of c_i includes in the cluster summary each term in $TopDistr(\alpha, \phi_i)$ with a probability of at least Pr_{ind}. Consider a document d which follows ϕ_i. For the preselection step to succeed, peer p needs to correctly identify from d at least one of the terms from $TopDistr(\alpha, \phi_i)$ that were also included in the cluster summary. In this case, the optimal cluster will be included in the list of clusters collected in the preselection step.

For a given value of $DocTF_{min}$ and for all $j \leq \alpha$, term $t_j[\phi_i]$ is correctly identified by peer p when the term frequency of $t_j[\phi_i]$ in d is at least equal to $DocTF_{min}$. The probability that this happens is denoted by $Pr[TF(t_j, d) \geq DocTF_{min}]$, and can be computed with Theorem 5.1. Therefore, the probability that the preselection step succeeds is:

$$Pr_{pre}(DocTF_{min}) \geq 1 - \prod_{j=1}^{\alpha} \left(1 - Pr[TF(t_j, d) \geq DocTF_{min}] \times Pr_{ind}\right) \quad (5.5)$$

Using Equation 5.5, peer p derives the maximum value of $DocTF_{min}$ that satisfies $Pr_{pre}(DocTF_{min}) \geq Pr_{pre}$. Then, it determines the document's terms with frequency at least equal to $DocTF_{min}$, and executes the preselection step.

5.5.5 Full comparison step

The preselection step for d returns a set of candidate clusters C_{pre}. From this list, peer p selects the clusters that will be fully compared with d, by using one of the filtering strategies introduced in Section 5.3.3. The conservative filtering strategy always returns the best cluster c_{opt} from C_{pre}, and thus it always assigns d to the best candidate cluster. For the Poisson-based strategy, we now obtain probabilistic guarantees that d will be assigned to c_{opt}. The Zipf-based strategy does not provide probabilistic guarantees for the full comparison step.

Our analysis follows the notation introduced in Section 5.3.3. The analysis assumes that for each term $t \in d$, the Probability Density Function (PDF) of the term scores of t in all clusters follows the Poisson distribution, where term scores are defined as the term frequencies normalized on the cluster length. In other words, $\{TF(t, c_1)/|c_1|, TF(t, c_2)/|c_2|, \ldots, TF(t, c_k)/|c_k|\}$ follow a Poisson distribution. Poisson distribution,

besides being a good fit for the term scores, has two useful properties: (i) Poisson fitting is efficient, as it only requires finding the average value, and, (ii) the convolution of k Poisson distributions with $\lambda_1, \ldots, \lambda_k$ is also a Poisson with $\lambda = \sum_{i=1}^{k} \lambda_i$. This second property is of particular importance, because it allows us to efficiently compute the PDF of $RCos(d, c)$ by convoluting the Poisson distributions corresponding to the normalized term frequencies of all document terms that are not found in $TopTerms(c)$.

To estimate the PDF of a term t, peer p finds the $AvgSc(t, C_{pre})$ value:

$$AvgSc(t, C_{pre}) = \sum_{c \in C_{pre}} \begin{cases} TF(t,c)/|c| & \text{, if } t \in TopTerms(c) \\ \frac{CluTF_{min}(c)-1}{|c|} & \text{, otherwise} \end{cases}$$

Since Poisson is a discrete distribution, for computing the distribution p discretizes the score range to m equidistant values, $v_0, v_1, \ldots, v_{m-1}$, with $v_i = 1 - i \times MaxSc(t)/m$. $MaxSc(t)$ refers to the upper bound score of t, which is $\frac{CluTF_{min}(c)-1}{|c|}$ given that $t \notin TopTerms(c)$. The value of m represents the resolution of the discretization.

The PDF of $RCos(d, c)$ for $c \in C_{pre}$ is estimated in a similar fashion. First, p finds all document terms that are not included in $TopTerms(c)$. These terms are the terms that may contribute to $RCos(d, c)$. The contribution of any of these terms to $RCos(d, c)$ is $\frac{TF(t,d)}{|d|} \times \frac{TF(t,c)}{|c|}$. Since the PDF of $\frac{TF(t,c)}{|c|}$ is a Poisson, the PDF of the term's contribution is also Poisson, with $\lambda_t = \frac{TF(t,d)}{|d|} \times AvgSc(t)$. The PDF of $RCos(d, c)$ is the convolution of the PDFs for all terms $t \in d \backslash TopTerms(c)$. We discretize the scores to m equidistant values, using $1 - PCos(d, c)$ as a maximum. The resulting PDF is:

$$Pr[RCos(d, c) \geq v_j] \leq \sum_{i=j}^{m} e^{-\lambda} \times \frac{\lambda^i}{i!} \tag{5.6}$$

where $\lambda = \sum_{t \in d \backslash TopTerms(c)} \lambda_t$. Note that the values of $AvgSc$ and $MaxSc$ are always upper bounds for the real average and maximum term scores. Therefore, $RCos$ is never underestimated because of this.

5.5.6 Algorithm Configuration

In the previous sections we have described PCP2P without focusing on how the user sets the parameters for the algorithm and what implications these parameters have. As in standard K-Means, a user can freely select the number of clusters k. In addition, she can configure the acceptable correctness probability $Pr_{correct}$ for her application. All other parameters are derived automatically, as follows.

First, a few sampled documents are collected from the network and are used to estimate the Zipf distribution skew. The algorithm computes the remaining values as follows. By default, α is set to 10 and Pr_{ind} to 0.9, since the algorithm adapts to these values to satisfy the probabilistic guarantees. The values of Pr_{pre} and Pr_{fcs} are set as follows. In conservative filtering, the full comparison step is always correct, therefore $Pr_{pre} = Pr_{correct}$ satisfies the probabilistic guarantees. For Zipf-based filtering we only provide probabilistic guarantees for the preselection step, and these are achieved by setting $Pr_{pre} = Pr_{correct}$. For Poisson-based filtering, $Pr_{pre} = Pr_{fcs} = \sqrt{Pr_{correct}}$ clearly satisfies the probabilistic guarantees. The algorithm then employs the analysis described above to derive the $DocTF_{min}$ and $CluTF_{min}$ that satisfy the desired probabilistic guarantees.

The previous combination of α and the probability values is not the only satisfying

combination. All combinations satisfying the constraint $Pr_{fcs} \times Pr_{pre} \geq Pr_{correct}$ satisfy the probabilistic guarantees, and the optimal combination of α, Pr_{ind}, Pr_{pre}, and Pr_{fcs} is the one that minimizes the cost. Preliminary experiments show that such an optimization could reduce the cost further by around 10%. Part of our future work is to efficiently identify the configuration that minimizes the cost.

5.6 Experimental Evaluation

In addition to the probabilistic analysis, PCP2P was evaluated experimentally with up to 1 million peers and 1 million documents. The experiments had the following objectives: (a) to evaluate quality and efficiency of PCP2P with different network configurations and multiple datasets, (b) to compare the algorithm with other P2P clustering algorithms, including the state of the art, as well as with the standard centralized counterpart, (c) to examine the scalability of PCP2P, with respect to network size, number of documents and number of clusters, and, (d) to assess the effects of load balancing. We now describe the evaluation methodology, datasets, and measures.

Evaluation measures. Efficiency was evaluated using the following criteria: (a) number of messages, (b) transfer volume, and, (c) number of document-cluster comparisons per clustering iteration.

Clustering quality was evaluated using the standard quality measures of *entropy*, *purity*, and *normalized mutual information* (*NMI*) [185, 109], which compare the clustering results against the human-generated document classification accompanying the datasets.

Given a clustering solution $C_{sol} = \{c_1, c_2, \ldots, c_k\}$ of N documents, and a set of human-generated classes $\mathcal{Z} = \{\zeta_1, \zeta_2, \ldots, \zeta_\tau\}$, such that each document belongs to exactly one class and exactly one cluster, the purity of C_{sol} is computed by labeling each cluster to the class with the most documents in the cluster, and then measuring the accuracy of the clustering. Precisely:

$$purity(C_{sol}, \mathcal{Z}) = \frac{1}{N} \sum_k \max_j |c_k \cap \zeta_j|$$

Purity takes values between 0 and 1, with high values denoting clusterings that closely match the human-generated classes.

Normalized mutual information is computed as follows:

$$NMI(C_{sol}, \mathcal{Z}) = -\frac{2\mathcal{I}(C_{sol}, \mathcal{Z})}{\sum_{j=1}^{k} \frac{|c_j|}{N} \log \frac{|c_j|}{N} + \sum_{j=1}^{\tau} \frac{|\zeta_j|}{N} \log \frac{|\zeta_j|}{N}}$$

where $|c_j|$ and $|\zeta_j|$ denote the number of documents in cluster c_j and class ζ_j respectively. $\mathcal{I}(C_{sol}, \mathcal{Z})$ is the mutual information, computed as follows:

$$\mathcal{I}(C_{sol}, \mathcal{Z}) = \sum_k \sum_j \frac{|c_k \cap \zeta_j|}{N} \log \frac{N|c_k \cap \zeta_j|}{|c_k||\zeta_j|}$$

NMI takes values in the range of 0 and 1, with higher values denoting better clustering solutions. Note that both purity and NMI never reach 1, if the number of clusters and

the number of classes are not equal. Therefore, the scores of two different clustering solutions are only comparable when they have an equal number of clusters.

Entropy is defined per cluster as follows:

$$E(c_j, \mathcal{Z}) = -\sum_{i=1}^{j} p_{ij} \log(p_{ij})$$

with p_{ij} the probability that a document of cluster j belongs to class ζ_i. The total entropy of the solution is computed by summing all normalized cluster entropies:

$$E(C_{sol}, \mathcal{Z}) = -\sum_{j=1}^{k} \frac{|c_j| \times E(c_j, \mathcal{Z})}{N}$$

with $|c_j|$ denoting the number of documents in cluster c_j. A low total entropy denotes a good clustering solution, with the caveat that a total entropy of 0 could be achieved by assigning each document to its own cluster.

Notice that computation of the three standard quality measures requires the reference classification $\mathcal{Z}$. Therefore, for choosing the datasets for the experiments, we required that these provided a reference classification.

Since PCP2P approximates the standard K-Means, as an approximation quality metric we measured the number of document assignments differing from the standard K-Means clustering solution. Precisely, if $C_{sol'}$ denotes the standard K-Means clustering solution, then the approximation quality is:

$$Ap(C_{sol}, C_{sol'}) = \frac{1}{N} \sum_{i} \max \arg_j |c_j \cap c'_i|$$

Clearly, approximation quality requires that the compared algorithms consider the same collection at every iteration. Therefore, we simulated churn such that all algorithms end up with an identical collection at each clustering iteration. Also, to exclude effects caused by the random cluster initialization, all algorithms were initialized with the same cluster centroids.

Datasets and Methodology. We simulated networks of up to one million peers and one million documents. The experiments were conducted on three datasets, a synthetic dataset that was generated according to the well-accepted Probabilistic Topic Model, and two real-world datasets, the Reuters Corpus Volume 1 (RCV1) [93], and MEDLINE [112], the largest standard text collections which include a human-generated classification. Note that a classification is required to compute the quality scores. RCV1 includes more than 800,000 categorized newswire articles, pre-processed with stopword filtering and stemming. MEDLINE on the other hand contains information for more than 11 million citations with abstracts, categorized according to the Medical Subject Headings vocabulary. To be able to apply standard quality measures, we have used a subset of the two collections, taking the articles and abstracts that belonged to exactly one class. This resulted in approximately 140 thousands articles for RCV1 categorized in 53 classes, and 130 thousands abstracts for MEDLINE, belonging to 40 classes. Furthermore, to systematically examine the effect of the collection's characteristics on the algorithm, and to evaluate it with a significantly larger dataset, we have also used synthetic document collections (SYNTH) with a size of 1.4 million

Experimental Configurations	
Network size	100 – 1,000,000; **100,000**
Number of documents	10000 – 1,000,000; **100,000**
Number of clusters	25 – 200; **100**
Probability Pr_{pre}	0.76 – 0.98; **0.9**

Table 5.2: Experimental configurations. The default values are emphasized.

documents each. These collections were created according to the Probabilistic Topic Model proposed in [157], from 200 composite language models, with different term distribution skews.

Unless otherwise noted, peer collections were created by partitioning the datasets uniformly to all peers. Churn was simulated by selecting a percentage of peers (up to 20% per clustering iteration), and replacing them with new peers, carrying new documents. The number of documents at each iteration, after churn, was 100,000 for the real-world datasets, and 1 million for the SYNTH dataset. Results are presented for 20% churn; the outcome was similar for other churn factors.

PCP2P was compared with two other distributed clustering algorithms: (a) *LSP2P*, which is the state of the art in P2P clustering [42], and, (b) *DKMeans*, a distribution of K-Means over P2P (see Section 5.1). Note that DKMeans produces exactly the same results as K-Means, and therefore constitutes a good comparison baseline. We have not included other P2P clustering algorithms in the comparison, e.g., [69, 47], since these do not scale to large networks, and to high-dimensional data like text, as discussed in Section 5.2.

PCP2P was also compared with a modified version of DKMeans, in which all peers downloaded all cluster centroids from cluster holders and performed the document-cluster comparisons locally. However, the results showed that this modified DKMeans is very inefficient regarding transfer volume compared to all other algorithms. For example, for the configuration of 10,000 peers and 50 clusters, the modified DKMeans requires 24,600 Mbytes compared to 3,770 Mbytes required by original DKMeans. This ratio becomes even less favorable with larger networks, and with more clusters. Therefore, we do not include the modified DKMeans in our discussion.

In the following, we present average results after 40 repetitions of each experiment. Notice that, owing to the continuous peer churn, the clustering algorithms never finish, i.e., the cluster centroids do not converge. This is also the case for real-world setups, with the network contents changing continuously. However, the quality results of all algorithms stabilize after about 15 iterations. Therefore, the algorithms are let to run for 20 iterations, before we measure their network cost and quality per iteration. The investigated configurations, and their default values are summarized in Table 5.2.

5.6.1 Quality

The quality of PCP2P is influenced by the correctness probability, the number of clusters, and the dataset characteristics (the number of documents, and the term distribution skew). The network characteristics, as well as the distribution of documents to peers, does not have any effect on the quality, since each document is clustered individually. Therefore, the results reported in this section are also representative of networks of different sizes, as well as of different document distribution models.

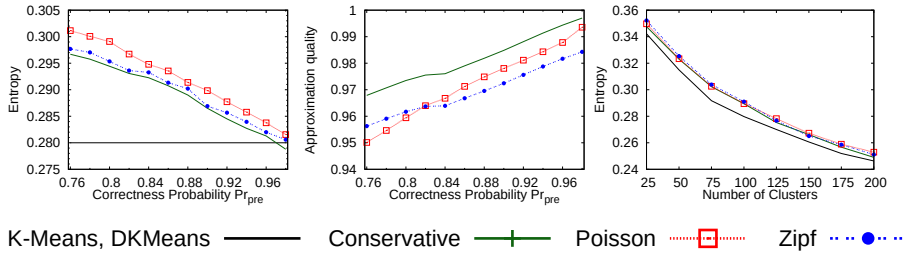

Figure 5.3: Quality: (a) Entropy, (b) Approximation quality, (c) Entropy for different number of clusters

Correctness probability. To examine the effect of the correctness probability Pr_{pre}, we executed PCP2P on a network of 10,000 peers, configured with Pr_{pre} in the range of 0.76 to 0.98. Figure 5.3(a) corresponds to the entropy measure for the RCV1 collection (lower entropy denotes better clustering quality). Table 5.3, Part I, summarizes the results for purity and NMI. As expected, the quality of all PCP2P variants increases with the correctness probability. Conservative PCP2P yields the best quality, since it never filters out potential optimal clusters. The Poisson and Zipf-based variants yield comparable quality. For probabilities higher than 0.9, all variants closely approximate DKMeans. The experimental results with MEDLINE and SYNTH had similar outcomes, summarized in Table 5.3, Part IV.

Figure 5.3(b) plots the approximation quality of PCP2P for different values of Pr_{pre}. Conservative filtering yields the best approximation quality, with a minimum of 0.97, and a maximum of 0.997. Zipf-based and Poisson-based PCP2P are also accurate, yielding an approximation quality of at least 0.95. We also note that the resulting approximation quality is always higher than the one expected by the probabilistic guarantees, since the guarantees compute upper bounds for the number of errors.

Number of clusters. As shown in Figure 5.3(c), the number of clusters also affects the quality of all algorithms. This is expected, considering that the computation of the quality measures depends on the number of clusters. However, the quality of PCP2P is always very close to the maximum quality delivered by DKMeans, even for the configuration with 200 clusters. Summarizing, the approximation quality of PCP2P does not decrease with an increase of the number of clusters.

Setup	Alg.	Quality Entr.	NMI	Pur.	Cost Msgs $\times 10^6$	Tr. Vol. (Gb)
I. Vary probability guarantees. RCV1, 100 clusters, 10,000 peers						
Pr_{pre}						
N/A	KMeans	0.279	0.524	0.689	186	13.2
0.8	Cons.	0.294	0.506	0.682	12.2	2.4
	Poisson	0.299	0.503	0.679	10.8	1.98
	Zipf	0.295	0.506	0.682	10.8	1.98
0.9	Cons.	0.286	0.516	0.689	13.7	2.7
	Poisson	0.289	0.513	0.687	12.1	2.24
	Zipf	0.286	0.516	0.689	12.1	2.23
0.98	Cons.	0.278	0.525	0.694	23.9	3.66
	Poisson	0.281	0.523	0.692	21.8	3.03
	Zipf	0.280	0.524	0.693	21.7	3
II. Vary #documents. SYNTH, exp.=1.0, Pr_{pre}=0.9, 100,000 peers						
Documents						
100000	KMeans	0.165	0.906	0.490	186.2	10.17
	Cons.	0.165	0.906	0.490	3.68	0.22
	Poisson	0.165	0.905	0.489	3.67	0.22
	Zipf	0.165	0.906	0.490	3.67	0.22
500000	KMeans	0.165	0.905	0.484	266.6	21.17
	Cons.	0.166	0.905	0.483	17.91	1.08
	Poisson	0.167	0.904	0.483	17.82	1.08
	Zipf	0.165	0.905	0.485	17.85	1.08
1000000	KMeans	0.164	0.907	0.484	367.1	34.91
	Cons.	0.163	0.907	0.485	34.78	2.08
	Poisson	0.164	0.906	0.485	34.61	2.07
	Zipf	0.165	0.905	0.482	34.67	2.07
III. Vary #peers. SYNTH, Pr_{pre}=0.9, 100 clusters						
Peers						
100000	KMeans	0.164	0.907	0.481	367.1	34.91
	Cons.	0.164	0.907	0.485	34.78	2.08
	Poisson	0.164	0.906	0.485	34.61	2.07
	Zipf	0.165	0.905	0.482	34.67	2.07
500000	KMeans	0.164	0.907	0.481	1147.58	69.80
	Cons.	0.164	0.907	0.485	41.38	2.42
	Poisson	0.164	0.906	0.485	41.27	2.44
	Zipf	0.165	0.905	0.482	41.29	2.44
Peers		*(III. continued...)*				
1000000	KMeans	0.164	0.907	0.481	2194.16	116.59
	Cons.	0.164	0.907	0.485	43.78	2.57
	Poisson	0.164	0.906	0.485	43.67	2.56
	Zipf	0.165	0.905	0.482	43.72	2.56
IV. Vary collection. Pr_{pre}=0.9, 100 clusters, 100,000 peers						
Collection						
MED	KMeans	0.557	0.222	0.373	186.2	12.58
LINE	Cons.	0.566	0.212	0.367	15.98	2.64
	Poisson	0.570	0.208	0.362	13.11	1.87
	Zipf	0.568	0.210	0.365	13.09	1.89
SYNTH	KMeans	0.170	0.899	0.491	186.2	11.81
exp.=0.5	Cons.	0.170	0.899	0.493	15.93	1.09
	Poisson	0.169	0.899	0.497	15.42	0.99
	Zipf	0.169	0.899	0.494	15.44	1
SYNTH	KMeans	0.165	0.906	0.490	186.2	10.17
exp.=1.0	Cons.	0.165	0.906	0.490	3.68	0.22
	Poisson	0.165	0.905	0.489	3.67	0.22
	Zipf	0.164	0.906	0.490	3.67	0.22
SYNTH	KMeans	0.164	0.907	0.487	186.2	9.47
exp.=1.4	Cons.	0.163	0.907	0.489	3.55	0.21
	Poisson	0.163	0.907	0.489	3.55	0.21
	Zipf	0.163	0.908	0.489	3.55	0.21
V. Vary #clusters. RCV1, Pr_{pre}=0.9, 100,000 peers						
Clusters						
50	KMeans	0.314	0.538	0.658	93.1	6.59
	Cons.	0.323	0.527	0.655	12.9	1.88
	Poisson	0.323	0.527	0.655	11.94	1.61
	Zipf	0.325	0.526	0.652	11.83	1.59
100	KMeans	0.279	0.525	0.689	186.2	13.17
	Cons.	0.289	0.513	0.685	13.67	2.69
	Poisson	0.289	0.514	0.686	12.1	2.24
	Zipf	0.291	0.512	0.683	12.5	2.23
200	KMeans	0.246	0.513	0.726	372.2	26.34
	Cons.	0.248	0.509	0.726	14.86	4.04
	Poisson	0.252	0.506	0.723	12.26	3.26
	Zipf	0.251	0.508	0.724	12.27	3.27

Table 5.3: Detailed experimental results

Dataset characteristics. To verify the applicability of the PCP2P for different text corpora, we have used the SYNTH collection, which enabled us to manipulate its characteristics in a systematic way. Particularly, all algorithms were executed on a network with a fixed number of peers, progressively increasing the number of documents up to 1 million. Table 5.3, Part II, presents key results of this experiment. Our first observation is that collection size does not have a significant effect on clustering quality; the minor fluctuations (i.e., between 0.164 and 0.168 for entropy) are due to the different document collections clustered at each configuration, and not due to an algorithmic factor. As such, the PCP2P quality always remains very close to the quality achieved by DKMeans, even for the largest collection with one million documents.

The term frequency distribution skew of the dataset is also an important factor for the quality of PCP2P, since the algorithm relies on the fact that term frequencies follow the Zipf distribution. Although it is widely accepted that document collections follow this distribution, different document collections exhibit different distribution skews [20]. For example, the RCV1 collection used in our experiments has a skew of 0.55 and MEDLINE has a skew of 0.59. Other values, typically around 1.0, are also frequently reported in the literature, e.g., in [20, 190]. To evaluate the influence of the skew on PCP2P, we have used the SYNTH collections generated with different Zipf skew factors, between 0.5 and 1.4. The example results, presented in Table 5.3, Part IV, confirm that PCP2P adapts to the collection skew and delivers high-quality clustering in all cases, very close to the DKMeans results.

5.6.2 Efficiency and scalability

We now investigate the influence on efficiency of the correctness probability, the network and collection characteristics, as well as the number of clusters. Furthermore, we examine the additional cost imposed by load balancing.

Correctness probability. Figures 5.4(a) and (b) show the number of messages and transfer volume per clustering iteration in correlation to Pr_{pre}. The results correspond to a network of 100,000 peers with the RCV1 collection, and with 20% churn. For comparison purposes the cost of DKMeans is also included in the plots. All PCP2P variants generate an order of magnitude less messages than DKMeans, with Poisson and Zipf-based PCP2P being the most efficient. Concerning transfer volume, conservative filtering requires between 17% and 27% of the transfer volume of DKMeans, whereas Zipf-based and Poisson-based filtering require between 15% and 22%. Note that the Poisson variant already provides 90% correctness probability with 6% messages and 17% of the transfer volume of DKMeans. The number of document-cluster comparisons – Figure 5.4(c) – which translates to computational cost for the cluster holders, is also substantially reduced compared to the baseline: the conservative strategy requires at most 12% of the DKMeans comparisons, whereas the Poisson and Zipf-based variants require below 1%, even for the highest investigated probabilistic guarantees, which deliver practically the same quality as DKMeans.

Network size. For evaluating the effect of the number of peers on PCP2P, we conducted experiments with networks of different sizes, and measured the cost for performing the clustering. In Figure 5.5(a) we present the results for the SYNTH collection, since this was the largest one and allowed us to simulate networks of up to one million peers. We see that the cost for PCP2P increases only logarithmically with network size.

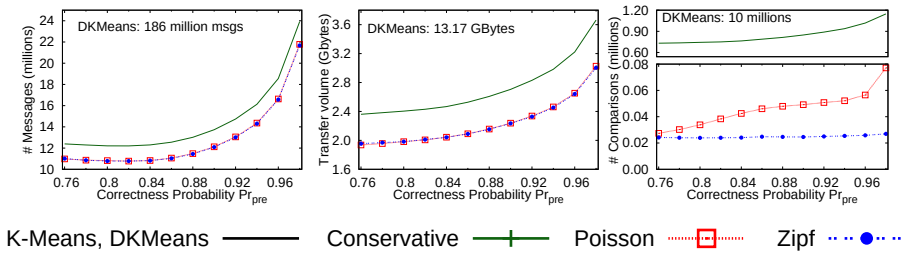

Figure 5.4: Efficiency: (a) number of messages, (b) Transfer volume, (c) number of of comparisons

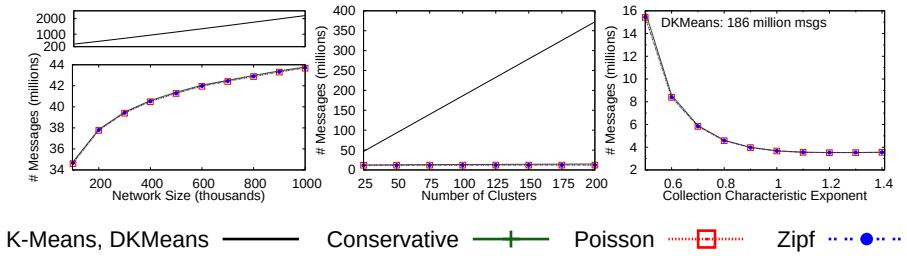

Figure 5.5: Number of messages: (a) varying network size, (b) varying number of clusters, (c) varying skew

This behavior is expected, and in accordance to the theoretical analysis (Section 5.4); the only factor changing with network size for PCP2P is the DHT access cost, which grows logarithmically. On the other hand, DKMeans cost increases linearly since each peer needs to communicate with all cluster holders, independent of the number of documents it carries. Similar results were observed on the other two collections.

Number of clusters. To examine the effect of the number of clusters to the efficiency of PCP2P, we repeated the clustering of the three collections on a network of 100,000 peers with up to 200 clusters. Figure 5.5(b) shows the number of messages in correlation to the number of clusters for the RCV1 collection. Clearly, all PCP2P variants scale favorably with the number of clusters. For example, for 25 clusters, the conservative variant induces 12 million messages, whereas for 200 clusters it causes only 3 million additional messages. The Poisson-based variant causes between 11.7 to 12.2 million messages in the same cluster range, whereas Zipf-based PCP2P causes 11.6 to 12.2 million messages. The same scale-up properties are observed with respect to the transfer volume and number of comparisons (Table 5.3, Part V). This essentially means that the network cost of PCP2P is only slightly affected by the number of clusters, making the algorithm scalable for a wide range of settings and requirements.

Dataset characteristics. Similar to the quality experiments, the SYNTH collection was used to evaluate the effects of the collection characteristics to the efficiency of PCP2P. Table 5.3, Part II, includes the results corresponding to different dataset sizes, for a fixed network size of 100,000 peers. As predicted by the theoretical analy-

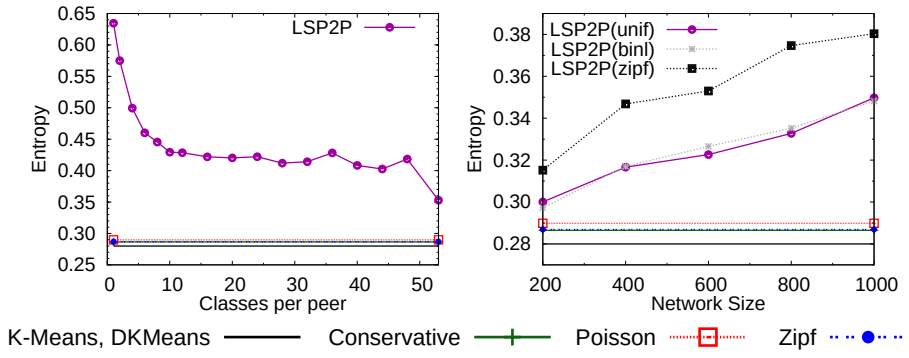

Figure 5.6: Quality of PCP2P and LSP2P: (a) varying the number of classes per peer, and, (b) varying the document distribution to peers

sis, PCP2P scales linearly with the collection size with respect to all cost measures. DKMeans also scales linearly to the number of documents, but incurs a significantly larger cost overall due to a larger constant factor.

Figure 5.5(c) displays the execution cost in number of messages, for a varying distribution skew, and for $Pr_{correct} = 0.9$. We see that for the same quality level, PCP2P cost is substantially reduced for higher skews. This behavior is expected: with higher skews, the first few most frequent terms of documents and clusters are sufficient for finding the candidate clusters, and PCP2P requires fewer comparisons for satisfying the probabilistic guarantees. For commonly reported skew values (around 1.0), the number of messages is reduced by two orders of magnitude compared to DKMeans. Nonetheless, even for a skew as low as 0.5, the cost of both PCP2P variants is already an order of magnitude lower than the DKMeans cost. Recall that the quality of the results remains unaffected by the skew factor (Section 5.6.1), since the algorithm adapts to the skew factor to satisfy the probabilistic guarantees.

Load balancing. As explained in Section 5.3.1, cluster holders may employ load balancing to avoid overloading. The load balancing process incurs a small network overhead for synchronizing the centroids between the cluster holders and their helpers. To examine this additional overhead, we have configured the load balancing threshold such that the load of each cluster holder never exceeds twice the average expected load, and repeated all experiments. The overhead in all examined configurations was less than 100 additional messages, and less than 10 Mbytes total. Therefore, load balancing does not impede the efficiency of PCP2P.

5.6.3 Comparison with other algorithms

PCP2P was also compared with LSP2P, the state of the art in P2P clustering. A preliminary testing of LSP2P with low-dimensional synthetic data verified the good results presented in [42], but also revealed a strong correlation of the algorithm's quality to the way the documents were distributed to the peers. Therefore, our further experiments focus on the effect of the document distribution to the quality of the compared algorithms.

Real-world peer collections are often multi-thematic, similar to real persons' interests. 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 reference classification accompanied with each dataset. Peers were creating their collections by: (a) randomly selecting i random categories/classes from the reference classification, and, (b) randomly selecting j documents for each class, ending up with $i \times j$ different documents.

We first examine the influence of the number of classes i assigned to each peer during the creation of the peer collections. Figure 5.6(a) plots the quality of the compared algorithms, when deployed on a network of 1000 peers for clustering the RCV1 collection. The number of clusters was set to 100, and PCP2P was configured with $Pr_{pre} = 0.9$. The quality of DKMeans and of all PCP2P variants is independent of the number of classes per peer, since these algorithms handle each document individually. On the other hand, the quality of LSP2P reduces with the decrease of i, and for $i \leq 10$, it becomes significantly worse than the optimal quality of DKMeans. In practice, this means that LSP2P requires that each participating peer carries documents from almost all classes to perform well. This is clearly a limiting factor for the applicability of LSP2P on text corpora, since it cannot be expected that real-world users have such a high variation of personal documents. In fact, in real-world networks, the number of possible classes and clusters might be even higher than the ones investigated here, aggravating the problem. The experiments with MEDLINE and SYNTH confirmed the same limitation of LSP2P.

For the second comparison, we compare the scalability of the compared algorithms by varying the number of peers in the network. To alleviate the pre-mentioned limitation of LSP2P that each peer requires a very diverse document collection, the documents were assigned to the peers randomly, completely ignoring the document classes. Therefore, each peer was expected to own a diverse set of documents. With respect to the number of documents per peer, we tested three different distributions: (a) a Zipf distribution, with skew equal to 1, (b) a binomial distribution, with average equal to $100,000/n$, where n denotes the number of peers, and, (c) a uniform distribution with the same average. These distributions are frequently considered in the literature for modeling the size of the peer collections.

Figure 5.6(b) plots the quality of the compared algorithms, for clustering the RCV1 collection to 100 clusters. We see that uneven document distributions, such as the Zipf distribution, create additional problems to LSP2P, which become more apparent for larger networks. The quality of PCP2P on the other hand remains unaffected from the documents distribution, as expected. Similar results were also observed with the MEDLINE and the SYNTH collection.

Concerning network cost, LSP2P was more efficient than PCP2P, since peers in LSP2P exchange cluster-granularity data instead of document-granularity data. However, the inability of LSP2P to deliver a quality clustering solution, and its requirement that all peers have documents from all classes makes the algorithm unsuitable for real-world scenarios.

5.7 Summary

We presented PCP2P, the first scalable P2P text clustering algorithm. PCP2P achieves a clustering quality comparable to standard K-Means, while reducing network cost by

an order of magnitude. We provided a probabilistic analysis for the correctness of the algorithm, and showed how it adapts to satisfy the required probabilistic guarantees.

Extensive experimental evaluation with up to 1 million peers and 1 million documents – real and synthetic – has verified the scalability and high effectiveness of all PCP2P variants, and its appropriateness for text collections with a wide range of characteristics. The results demonstrate that PCP2P quality closely approximates the optimal clustering quality of K-Means and DKMeans as predicted by the theoretical analysis, with a small fraction of the cost. The network benefits often exceed one order of magnitude compared to DKMeans, and a configuration mechanism based on probabilities enables fine-tuning of the cost/quality tradeoff of PCP2P. Finally, the experiments have also shown that PCP2P outperforms LSP2P, the current state of the art algorithm, and addresses its scalability limitations.

Chapter 6

P2P Text Classification

Automatically structuring heterogeneous document collections into thematically coherent subsets is a relevant task for a variety of Web applications. Some frequent examples are focused crawling, structuring Web directories, social bookmarking, and email spam filtering [72, 8, 129, 29]. For these classification applications, supervised learning has become the method of choice. Supervised machine learning approaches employ the following methodology: they use a set of training items manually assigned to categories by the users, to build classifiers for automatic assignment of category labels. A sufficient amount of training data is crucial for achieving high classification accuracy. However, labeling enough data and keeping the training set sufficiently diverse and up-to-date can require a substantial manual effort.

Collaborative solutions, where users combine information and resources, have recently gained importance in various areas, e.g., collaborative semantic desktop [119], collaborative tagging [170], sharing of e-mail signatures for spam detection [84], P2P telephony, e.g., Skype, and video streaming [153]. Collaborative solutions have been also explored for classification, to address the issue of insufficient training data [165, 115, 102, 6, 152]. The basic idea of collaborative classification is to aggregate information from different users in a collaborative network, for constructing better machine learning models that can be used by every network member for their individual information demands. A naive approach based on sharing training samples directly among users to obtain larger training sets is prohibitive, since it ignores privacy, security, and copyright aspects of the user's personal information sources, and also leads to high network costs for the participants.

For distributed classification, Luo et al. [102] introduce a voting-based algorithm for P2P networks, where voting vectors with components corresponding to vote counts for categories are maintained in a distributed manner. However, their setting implies that personal (test) data to be classified have to be either available on all peers or need to be propagated. In [6] only a selected subset of training vectors, namely the support vectors obtained from locally built RSVM models (a modification of support vector machines) are propagated in the network; this helps to overcome some of the privacy and network cost issues, however, only to a certain extent.

Our approach is based on exchange of classification models instead of training or test data, as also proposed in [152]. Each node combines classification models received from other nodes with its own model to build a meta classifier, which is then used for organizing the user's individual document collection. Compared to typical, standalone classifiers, meta classifiers do not suffer from a cold start problem. They are

also substantially more accurate, since they correspond to a much larger training set. As model dissemination infrastructure, we use an unstructured peer-to-peer network. For instance, participating nodes might use a plug-in for folders in their file system for enabling collaborative document organization, or an email client plug-in for enabling collaborative spam filtering. The infrastructure is fault-tolerant, and does not require central coordination.

For efficient model exchange, it is crucial to keep the models compact. However, the number of model components (dimensions) and, thus, the size of the models can become very large, leading to considerable communication costs. To drastically reduce these costs and enable a fine-grained and flexible control of the cost/quality tradeoff, we combine meta-classification with model dimensionality reduction. Instead of exchanging the complete models, participating nodes only distribute the most important components of their models. This approach enables participating users to build highly accurate meta models with negligible network cost.

We validate our approach on four large standard collections for text classification, and compare it to existing state of the art methods. Our experiments evaluate the influence of model propagation and model dimensionality reduction on classification accuracy and distribution efficiency, and identify sweet spots for the corresponding parameters. The results show that the computation and communication resources required for participating in the network are negligible, and therefore documents can be classified in real-time, even on commodity machines.

The rest of this chapter is organized as follows. In the next section we summarize and compare related work on collaborative classification. In Section 6.2 we present our approach in detail, covering the classification, distribution of local models, dimensionality reduction, and the construction of meta classifiers. Section 6.3 shows the results of our large-scale evaluation with respect to classification accuracy and efficiency, as well as comparisons with existing approaches. We conclude the chapter in Section 6.4.

6.1 Related Work

The machine learning literature has studied a variety of ensemble based meta methods such as bagging, stacking, or boosting [23, 57, 87, 99, 169], and also combinations of heterogeneous learners, e.g., [182]. For bagging, an ensemble consists of classifiers built on bootstrap replicates of the training set. The classifiers' outputs are combined by plurality vote. For stacking, multiple classifiers are trained on parts of the training set and evaluated on the remaining training documents. The outputs of the classifiers are used as feature values for training a new classifier (stacked generalization). Boosting can be viewed as a model averaging method. Here a succession of models is built, each one trained on a data set in which the points misclassified by the previous model are given more weight. However, these approaches do not deal with distributed settings.

Mladenić et al. [120] describe a pruning method for normal vectors of Support Vector Machines (SVMs), which they use for feature selection to make the training more efficient. Our work is the first to apply this technique in the context of efficient model propagation in a distributed environment. Another technique for dimensionality reduction was proposed in [8], for the purpose of reducing memory requirements of the local classification process. This technique uses feature hashing to collapse several features into one dimension. As we show in Section 6.3, feature hashing can substantially degrade classification quality.

Distributed classification can be characterized according to distribution infrastruc-

ture and meta-classification approach. With respect to distribution, centrally coordinated [165], hierarchical [12] and purely decentralized P2P classification algorithms have been proposed. The latter are realized either using a distributed hash table [7], or an unstructured topology [115, 102, 6, 152]. These approaches employ various meta-classification techniques. In [165] the authors show how to compute a distributed Naive Bayes classifier with centralized coordination. In [115] the parameters of local generative models are transmitted to a central site and combined. Luo et al. [102] introduce a voting-based classification algorithm in a P2P network. They compute voting vectors with components corresponding to vote counts for categories in a distributed manner. Their setting substantially differs from ours in that test data in their scenario either have to be available on all peers or have to be propagated (instead of exchanging classification models). This is prohibitive in applications such as spam filtering, both in terms of network cost for large test sets of emails, and due to privacy reasons. This issue affects all approaches requiring distribution of training or test sets. Cascade RSVM [6] applies Reduced SVM [92] to distributed classification. A variant of Cascade RSVM, which uses a DHT as underlying topology has been published in [7]. Instead of sharing complete training sets, peers exchange only their support vectors. In a cascaded classification process, the peers add the received vectors to their respective training set and re-classify, until the process converges. Support vectors directly represent individual local documents, raising privacy issues with respect to their propagation. In Section 6.3, we show that our approach clearly outperforms Cascade RSVM.

Our notion of distributed model sharing and propagation in a P2P environment is closest to the one described in [152]. However, that work treats classification effectiveness aspects only, not considering the incurred communication costs. Because complete models are exchanged between nodes, these costs can become unacceptable in the case of high-dimensional data such as text. In this chapter, we show how this approach can be made practical by compacting the transferred models and limiting the number of exchanges in such a way that communication cost becomes very low while classification quality remains nearly unaffected.

The proposed collaborative classification algorithm belongs to the class of *local* algorithms, i.e., each node only needs to cooperate with a small set of nearby neighbors to perform the desired tasks. In recent years, local algorithms for various other data mining problems have been developed, e.g., computing an average [113], conducting a majority vote [148], or k-means clustering [42]. See [40] for a survey. Local algorithms scale extremely well, and are very robust because any failure only affects a small neighborhood.

To the best of our knowledge, the work described in this chapter is the first to apply flexible dimensionality reduction for efficient classification model propagation and combination, and to provide a thorough experimental study of the resulting efficiency-effectiveness tradeoffs in the context of distributed and collaborative classification.

6.2 Collaborative Classification with CSVM

We now describe our framework for distributed execution of linear discriminative classification. CSVM, short for Collaborative SVM, combines local classification, model sharing, and dimensionality reduction, to realize scalable distributed classification with an excellent quality/network cost tradeoff. We describe the framework assuming that the local classifiers are built using support vector machines (hence the name CSVM),

but the framework is in fact agnostic to the used local classifiers. For example, it can also use Reduced SVM [92], which increases training efficiency for large training sets by choosing a suitable subset of the training data. It is also applicable to other linear discriminative classification approaches, e.g., Fisher's Discriminant [54].

In CSVM, nodes performing classification are connected in a peer-to-peer network. We assume that each peer in the network has its own training set. The algorithm consists of the following steps:

- Every peer computes a local classification model using its own training set.
- Peers reduce their local models, and exchange them with a small number of selected neighbors.
- Each peer merges the received models with its own model to construct a more powerful meta classifier, taking reliability weights into account.

This process is repeated periodically to account for changes in the network, and to incorporate new training documents. The resulting meta classifiers exhibit a much higher quality than the local ones, and can be used at each node for classification purposes. In the following, we describe the elements of the algorithm in detail.

Classification with Support Vector Machines. In SVM classification, the data to be classified is assumed to be in the form of feature vectors. For example, a feature vector of a document might consist of the frequencies of the terms occurring in the text, taking into account the inverse document frequency for weighting. In the following, we focus on *binary* classification, i.e., the classifier needs to distinguish between two classes of items, usually labeled *positive* and *negative* instances. There exist various techniques to reduce multi-class classification to a set of binary classification problems that can be solved separately [5].

SVMs, as all linear discriminative algorithms, construct a hyperplane as classification model, described by the equation $\vec{w} \cdot \vec{x} + b = 0$, where $\vec{w}$ is the normal vector and b the bias of the hyperplane. The constructed hyperplane separates the set of positive training examples from the set of negative examples with maximum margin. This training requires solving a quadratic optimization problem whose empirical performance is somewhere between quadratic and cubic in the number of training documents [25]. Given an SVM model m, for classifying a new, previously unseen item e with feature vector $\vec{e}$, we only need to test whether this vector lies on the positive side or the negative side of the separating hyperplane. This decision simply requires computing the scalar product of $\vec{w}$ and $\vec{e}$, and results in a classification score $s(e, m)$, which can be positive or negative, and corresponds to our confidence of e being positive resp. negative. SVMs have been shown to perform very well for a wide spectrum of applications, e.g., text classification [46], spam detection [149], content-based image retrieval [161], speech recognition [59], and medical analysis [91].

Model Propagation in the Network. In our framework we are given a set of peers $P = \{p_1, p_2, \ldots, p_l\}$. Peers form an unstructured P2P network, as the ones described in Section 2.2, where each peer chooses its neighbors randomly. Techniques to construct and maintain such networks are well studied, e.g., [30, 162, 66, 163], and any of these techniques can be applied to maintain the P2P infrastructure. The resulting random graph network topology can be described with the neighborhood relation $N \subset P \times P$. Note that our algorithm only requires the availability of a minimum amount of

neighborhood links for each peer, i.e., at least n neighbors. This requirement can also be satisfied from structured P2P topologies, e.g., Chord (cf. Section 2.4), thus CSVM can also be built on top of a structured P2P network.

Each peer $p_i \in P$ maintains its own item collection C_i and training set $T_i \subset C_i$, with $|T_i| << |C_i|$. Every item in the training set T_i is labeled as either positive or negative. A peer p_i with a set of neighbors $N(p_i)$ builds and propagates its local SVM model m_i as follows. First, it uses its training set T_i to construct m_i. It then randomly chooses n of its neighbors, denoted as $R(p_i)$, and requests their local classification models. Using the models received from these peers, it computes a new meta model, as described below. This process is repeated periodically, to take into consideration the new training documents accumulated at the collaborating peers.

Dimensionality Reduction. The propagation of the SVM models incurs a network cost for the participating peers. This cost is determined by the dimensionality of the models, i.e., the size of the normal vector $\vec{w}$ of each model. As each feature introduces an additional dimension in $\vec{w}$, these feature vectors can become very large and their transmission costly. To address the efficiency in the context of centralized classification, Mladenić et al. [120] describe a pruning method for normal vectors of Support Vector Machines (SVMs). They show that components in the normal vector $\vec{w}$ with high absolute values are the most important ones for the classification model.

We exploit this result in a novel way for obtaining a much more compact model representation without significant loss of information. Particularly, each peer determines its top-k normal vector dimensions with highest absolute values, and discards the remaining dimensions. In the following, we denote the reduced normal vector of peer p_i as $\vec{w'}_i$. In Section 6.3 we study the influence of parameter k on the network load and on the accuracy of the classification, and show that the combination of highly reduced model representations still yields highly accurate meta models. Our evaluation shows that the employed approach incurs negligible quality loss, with a very low network cost. Note that our system is not limited to this dimensionality reduction technique; there is a plethora of alternative dimensionality reduction methods that could also be employed, such as the ones described in [55, 181].

Meta Model Construction. We now describe how a peer p_i combines the set of models received from its neighbors and its own model into a single meta model $meta_i$. For clarity, we first explain how the meta model is constructed assuming that peers exchange unreduced classification models, and we then consider the case of exchanging reduced models. A local linear discriminative model m contains the hyperplane representation, i.e., a tuple $< \vec{w}, b >$. When transmitting m, a vector $\vec{l}$ is added which maps the dimensions of $\vec{w}$ to features, i.e., the ith component of $\vec{l}$ relates the ith component of $\vec{w}$ to its corresponding feature. This eliminates the need to maintain a common feature enumeration over the P2P network.

We note that some information can be inferred from the exchanged classification models about the word distribution of the user's documents. The exchange of a certain amount of user information is unavoidable for collaboration. However the reduced models can be seen as a very compressed statistical representation of the data, and, thus, reveal much less information than the complete training data, or the support vectors.

Let $M_i = \{m_i\} \cup \{m_j : p_j \in R(p_i)\}$ denote the set of all models available at peer p_i, i.e., its own model and the models requested from its neighbors. To classify an item e with feature vector $\vec{e}$, a peer combines classification scores $s(e, m_j)$ of the in-

dividual models in $m_j \in M_i$ to a meta score. For this combination, each model m_j is assigned a weight r_i according to the respective model's reliability. In our experiments (see Section 6.3) we use the respective training set size as reliability weight. More complex weights could take into account other factors, such as trust [77]. Assuming that the reliability weights are normalized, the meta score is computed as $s(e, meta_i) = \frac{1}{|M_i|} \sum_{m_j \in M_i} r_j \cdot s(e, m_j)$.

Merging of the individual scores is equivalent to computing a single hyperplane $\vec{w}_{meta_i} \cdot \vec{x} + b_{meta_i}$, where $\vec{w}_{meta_i}$ is the combined normal vector and b_{meta_i} the combined bias. The combined normal vector is computed as $\vec{w}_{meta_i} = \frac{1}{|M_i|} \sum_{m_j \in M_i} r_j \cdot \vec{w}_j$, where $\vec{w}_j$ is the normal vector of model m_j. Note here that $\sum$ denotes the sum of the corresponding vectors $\{\vec{w}_j : m_j \in M_i\}$, respecting the mapping of features to vector dimensions defined by $\vec{l}$. The combined bias is obtained similarly as $b_{meta_i} = \frac{1}{|M_i|} \sum_{m_j \in M_i} r_j \cdot b_j$. Combining all models to a single meta model per peer is desirable for efficiency reasons. In particular, the computational cost for classifying an item with feature vector $\vec{e}$, using $|M_i|$ different models is $O(|M_i| \times |\vec{e}|)$, while the cost for evaluating it using the meta classifier, as we do, is only $O(|\vec{e}|)$.

As explained, peers in our system reduce the models to save network resources. In practice, each peer only transmits the reduced normal vector $\vec{w'}$, the corresponding encodings $\vec{l'}$, and bias b. The actual meta model hyperplane constructed at each peer p_i is $\vec{w'}_{meta_i} = \frac{1}{|M_i|} \sum_{m_j \in M_i} r_j \cdot \vec{w'_j}$. The classification process remains the same as for the unreduced case.

Cost Model. CSVM belongs to the class of local algorithms, since the communication cost for each peer to build the meta model is independent of the size of the network. This cost depends on the number of neighbors $n = |R(p_i)|$ of each peer p_i, and on k, the number of the top model components exchanged. The network cost for constructing each meta model is $C_{meta} = O(n \times k)$, and the total communication cost for one period in a network of $|P|$ peers is $O(|P| \times n \times k)$. In Section 6.3.4 we discuss how to optimize the parameters k and n for a given network cost constraint to maximize the accuracy of the meta classifier.

6.3 Experimental Evaluation

The experimental evaluation had the following objectives:

- The evaluation of scalability and effectiveness of CSVM in dependence of its system parameters, number of neighbors per peer (Section 6.3.2), and number of dimensions per model (Section 6.3.3). This includes a discussion on the parameter tuning for the algorithm, i.e., the influence of optimized parameter selection for a given network cost budget (Section 6.3.4).

- The validation of CSVM performance characteristics using different real-world datasets, and using different sizes of training data per peer (Section 6.3.5).

- The comparison of CSVM with the state of the art in distributed and collaborative classification (Section 6.3.6).

We start by describing the experimental setup.

6.3.1 Experimental Setup

As described in Section 6.2, the CSVM framework can employ different linear discriminative classifiers for computing the local models. In our experiments, we tested CSVM using standard SVMs (denoted as *CSVM*) as well as Reduced SVM classifiers [92] (*CRSVM*). To investigate the benefits of collaborative classification, we compared these two CSVM variants with their non-collaborative counterparts (denoted as *SVM* and *RSVM*) where each peer uses only its local classifier built solely on its own training set. As a gold standard for collaborative classification, we also show the quality of non-distributed SVM classification (*CENTR*), computed on the union of the training sets of all peers. Notice that the CENTR classifier is only theoretical; as we explained already in the introduction, it has important scalability and privacy issues for large P2P networks. We also compared our approach with the state of the art P2P classifier, Cascade RSVM [6] (*CASC*). Finally, to examine the quality of our dimensionality reduction, we compared our approach with [8] (denoted henceforth as *HASH*), a state of the art non-distributed approach to reduce the feature space for SVM classification. We examined the performance of two HASH variants, the standard one which uses standard SVM for computing the classifiers, and a second one using RSVM (*RHASH*). All algorithms employ the LIBSVM implementations of SVM and RSVM [97].

In order to analyze the effect of the system configuration on the efficiency and effectiveness of CSVM, we conducted a wide range of experiments, varying the number of collaborating neighbors, and the dimensionality of the shared models. All experiments were repeated with different training set sizes per peer, and with four different datasets, to verify the applicability of CSVM to distinct usage scenarios. In the following, we report on experiments for a network of 100 peers built over an unstructured P2P network. Note that the algorithm's accuracy and efficiency for the participating peers is independent of both network size and network topology. Therefore, all results also apply to larger and differently formed P2P networks.

The experiments were conducted on four standard, Web-based datasets [97, 32]:

rcv1: The Reuters Corpus Volume 1 dataset consists of more than 800,000 news feed articles, and contains about 47,000 features.

trec: The TREC 2007 spam corpus, consisting of 75,000 emails, manually assessed as spam or ham. The dataset has a total of 395,000 features.

news20: The 20 Newsgroups dataset consists of approximately 20,000 newsgroup articles with 1,3 million features.

realsim: Approximately 72,000 UseNet articles from four discussion groups in the topics of 'simulated auto racing', 'simulated aviation', 'real autos', and 'real aviation'. The classification objective is to separate the documents about simulation from the others. This dataset has a total of 21,000 features.

We used the standard features and ground truth for binary classification available from [97] for the rcv1, news20, and realism datasets. For the trec collection we used term-based TF-IDF features, and the user assessments in [32] as ground truth. Each dataset was split into a training set and a disjoint test set. From the training set, we assigned to each peer a local training collection. Unless otherwise noted, each local training collection consisted of 25 positive and 25 negative randomly selected documents. To increase the reliability of our quality assessments, all peers were evaluated on the full test set; note that this does not influence the integrity of the evaluation since the peers do not exchange any information on the test data.

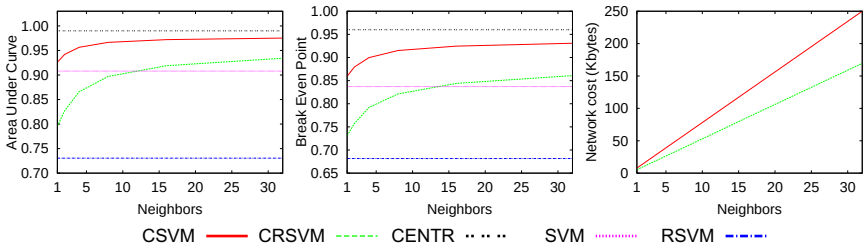

Figure 6.1: Influence of the number of neighbors on classification quality and cost (a) Area Under Curve, (b) Precision-Recall Break Even point, (c) Network cost per peer.

Evaluation Measures. As efficiency measures, we have used the network cost per peer as well as the processing cost required for classifying each document. Effectiveness was evaluated using the standard classification quality measures: (a) the Receiver Operating Characteristic (ROC) Curve [53] as well as its aggregate measure, the Area Under the ROC Curve (AUC), and (b) the Precision-Recall Break Even Point (BEP). AUC and BEP values close to 1 indicate highly accurate classifiers, whereas values close to 0 correspond to low classification quality. ROC curves are used for visualizing the performance of binary classifiers, and show the ratio of misclassified items in relation to the ratio of correctly classified items. In the literature, these two measures are widely used for evaluating binary classification scenarios, e.g., [33, 49, 58, 18, 32].

6.3.2 Influence of the Number of Neighbors

We first examine how the number of neighbors per peer influences the performance of CSVM. Here, *number of neighbors* refers to the number of peers that each peer exchanges models with, i.e., the cardinality of $R(\cdot)$ (cf. Section 6.2).

Quality. Figures 6.1(a) and (b) show the AUC and BEP measures, respectively, for CSVM and CRSVM configured with different neighborhood sizes. The results correspond to the rcv1 collection, with the number of dimensions reduced to 500. For comparison, we also include the performance of SVM and RSVM, the two non-collaborative approaches. For illustration purposes, SVM and RSVM are plotted as horizontal lines; the number of neighbors for both algorithms is 0 by definition, though.

Both CSVM and CRSVM clearly outperform the corresponding non-collaborative approaches. As expected, the benefit of collaboration increases with the neighborhood size. While, for instance, with just one neighbor per peer CSVM and CRSVM perform only marginally better than their corresponding non-collaborative counterparts, with 8 neighbors, we can already observe an improvement of more than 7% compared to the corresponding baselines. We also observe that the approaches using standard SVM (i.e., SVM and CSVM) achieve higher quality than the ones employing RSVM. The reason is that RSVM trades classification quality for speed, resulting in less accurate classifiers than standard SVM.

CSVM and CRSVM achieve significant improvements compared to the standard SVMs, even for small neighborhood sizes. In particular, CRSVM with just 4 neighbors yields a performance improvement of more than 10% compared to RSVM. Similarly, CSVM with 4 neighbors achieves a performance increase of more than 5% compared

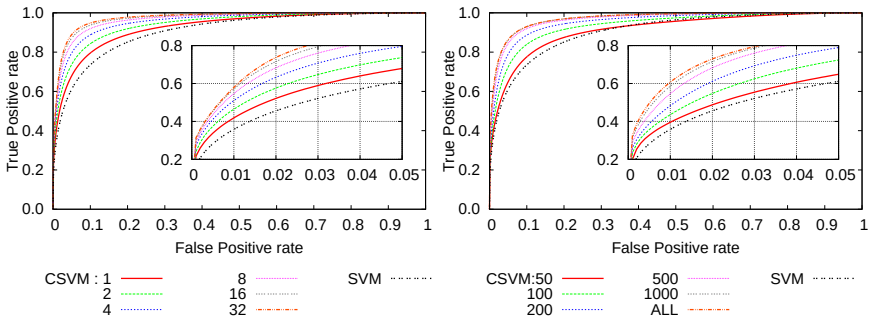

Figure 6.2: ROC curves for different neighborhood sizes and number of dimensions

to SVM. Adding more neighbors per peer further improves classification quality at a slower rate.

The ROC curve for the CSVM experiments (Figure 6.2(a)) reveals further insights on the strengths of CSVM. The improvement of both algorithms is particularly apparent on the left hand side of the curves, i.e., with false positive rates less than 0.05. This is generally the most interesting area for classification scenarios such as collaborative spam filtering, which involve a high cost for false positives. The ROC curves also confirm our previous observation that small neighborhood sizes are sufficient for achieving significant improvements; for example, the ROC curve corresponding to 8 neighbors already closely approximates the one of 32 neighbors. Similar results apply to CRSVM (see Figure 6.3(a)).

Efficiency. In Figure 6.1(c), we show how network cost develops when increasing the number of neighbors. Note that the corresponding network cost for non-collaborative SVM and RSVM is 0, since these do not employ model exchange. As expected (cf. Section 6.2), the network cost of both CSVM and CRSVM is linear with the number of neighbors. Due to the compactness of the SVM models after dimensionality reduction, the network cost per peer is well-manageable, even for deployment over mobile networks. For example, the total network cost per peer for the setup with 32 neighbors (the maximum value considered) is only 250 Kbytes for CSVM and only 170 Kbytes for

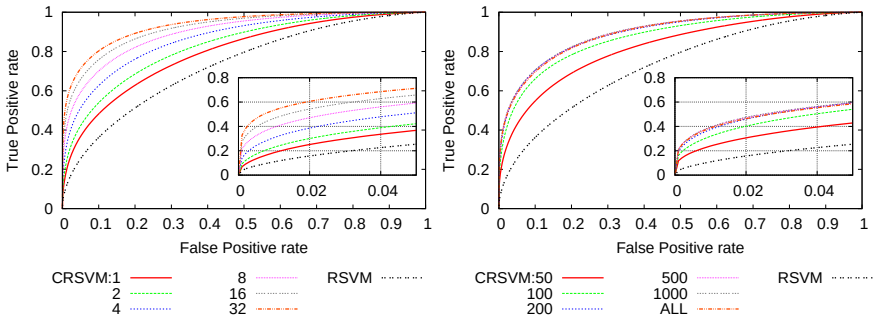

Figure 6.3: ROC curves for CRSVM: (a) number of neighbors, (b) number of dimensions

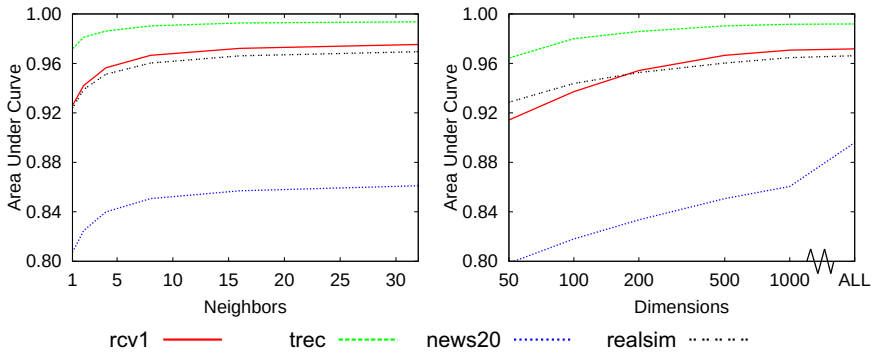

Figure 6.4: Influence of (a) number of neighbors, (b) number of dimensions, to quality, for all datasets

CRSVM. The lower network cost of CRSVM compared to CSVM is due to a specific property of RSVM: it generates more compact models than standard SVM. In some instances, these models have even less than the maximum allowed number of dimensions, i.e., less than 500 in our configuration. Therefore, the network cost of CRSVM is sometimes even lower than the expected cost.

With respect to computational complexity, all compared algorithms require the same time for classifying a document, approximately 0.01 milliseconds on a single AMD 2.7 GHz processor. Classification cost is linear to the number of non-zero components of the document vector, i.e., the number of distinct terms; therefore, the number of neighbors per peer, as well as the number of dimensions, does not influence the computational complexity of CSVM. Classification time is negligible, making the algorithm suitable for online classification. The periodic merging of the models also takes a negligible amount of time, less than 10 milliseconds per peer, even for the case where all dimensions are kept.

The qualitative results of the experiments on the other datasets are similar, as can be seen from the AUC values shown in Figure 6.4(a). Summarizing, the first set of experiments shows that increasing the number of neighbors leads to better classification quality, and that a small number of neighbors already yields substantial improvements compared to the baselines, with negligible network and computational overhead.

6.3.3 Influence of the Number of Dimensions

In our second experimental series we examined the influence of the number of model dimensions on the performance of CSVM and CRSVM. Figures 6.5(a) and (b) present the AUC resp. BEP values for CSVM and CRSVM, configured with different numbers of dimensions (ranging from 50 to all dimensions). The results are shown for the rcv1 collection, with 8 neighbors per peer. As before, we include the non-collaborative counterparts for comparison.

Quality. As expected, increasing the dimensionality has a positive effect on the classification quality. For instance, increasing from 50 to 200 dimensions increases the AUC value from 0.914 to 0.954. Also, similar to the case of increasing the neighborhood size, the number of dimensions does not need to be very high for substantial quality improvements; both CSVM and CRSVM yield already substantial benefits

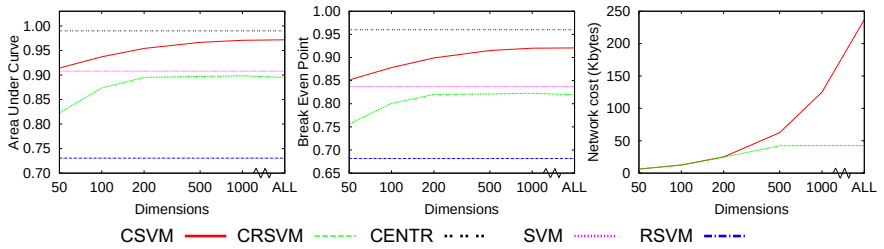

Figure 6.5: Influence of the number of dimensions on classification quality and cost (a) Area Under Curve, (b) Precision-Recall Break Even point, (c) Network cost per peer.

with 500 dimensions, achieving a classification quality almost equal to the unreduced models (denoted with ALL on the X axis). The ROC curves further confirm these observations (Figure 6.2(b) and Figure 6.3(b)).

The results on the other three datasets are summarized in Figure 6.4(b). An interesting observation is that the news20 collection benefits more from the increase in the number of dimensions than the other datasets. In particular, we observe substantial quality increase in the news20 results, even after increasing from 1000 to all dimensions; for the other datasets, this change yields just a very small additional performance. This is due to the larger number of features contained in news20, namely 1.3 million compared to less than 400,000 for the other datasets. This result also yields an interesting challenge, for estimating the number of important dimensions for each dataset, which we plan to address in our future work.

Efficiency. We observe that network cost grows linearly with the number of model dimensions (Figure 6.5(c)). For the configuration with 500 dimensions, which yields a near-optimal classification, the network cost reaches a maximum of 62 Kbytes per peer. Interestingly, the cost for the CRSVM approach reaches a plateau after 500 dimensions. This is because in most cases, RSVM generates local models of less than 500 dimensions per peer. Therefore, the dimension limit does not lead to further model reduction. This is also the reason why CRSVM with 500 dimensions achieves the same quality as CRSVM with unreduced models.

As explained in Section 6.3.2, computational complexity is orthogonal to the number of neighbors and the number of model dimensions; therefore, the classification time of CRSVM is equal to the one of the standard, non-collaborative SVM algorithm.

6.3.4 Parameter Tuning

As demonstrated in the previous experiments, the accuracy of the algorithm is controlled by the number of collaborating neighbors and model dimensions. These parameters can be tuned to optimize classification accuracy for a given, user- or system-defined, network cost quota.

In order to explore the optimal combination of number of neighbors and dimensions we tested both algorithms with different quotas. These were expressed as the maximum transfer volume per peer participating in the network (including both incoming and outgoing transfer volume). For each quota, we executed all possible setups – combinations of the number of neighbors and dimensions – and identified the combination resulting in the maximum AUC value. Figure 6.6 summarizes the experimental

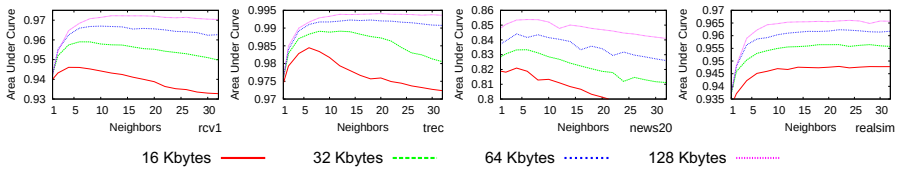

Figure 6.6: Classification quality for various network constraints: (a) rcv1, (b) trec, (c) news20, (d) realsim.

results for the four collections. The X axis depicts the neighborhood size whereas Y depicts the AUC measure. We see that the AUC measure is fairly stable in the area between 5 and 20 neighbors per peer, having a maximum difference of less than 0.02. In particular, a default value of 8 neighbors per peer provides already near-optimal results for all examined configurations (more than 99% of the optimal AUC). Interestingly, the function of AUC with respect to the two control parameters is always convex, which can facilitate the efficient optimization of the parameters using, for instance, convex optimization techniques. Part of our future work will focus on enabling the peers to adapt dynamically and efficiently to their network limitations for maximizing the classifier performance.

6.3.5 Influence of Training Data Characteristics

The previous experiments considered only scenarios where the training data is assigned uniformly to the peers. However, CSVM can also be applied to P2P networks with heterogeneous training set sizes, corresponding, for instance, to scenarios where the effort put into creating local training sets varies strongly between users. In fact, these are precisely the setups where the non-collaborative algorithms fail, due to insufficient training data on some peers (the cold start problem). Therefore, we examined two alternative distributions characterizing the number of documents per peer, (a) Poisson, and, (b) Zipf distribution. To allow for comparison with the previous results we kept the average number of documents per peer at 50. As before, we conducted two series of experiments, by varying either the number of neighbors or the number of dimensions.

Figure 6.7 presents the AUC measures for the four datasets, and the discussed distributions. We see that the AUC values are practically equal for each dataset, independent of the distribution of the training set. This means that the quality of CSVM stays unaffected of the distribution of the training sets. Even for the Zipf distribution, where most of the peers have a very low number of training data, CSVM still yields practically the same results by combining weighed models from neighboring peers. CRSVM (Table 6.1) is more influenced by the distribution, but the derived quality is still acceptable, and always significantly better than the centralized counterpart. The same qualitative results were achieved with respect to the BEP measure.

With respect to the two non-collaborative algorithms, SVM and RSVM, we observe that the distribution has a strong negative effect. Table 6.1 shows the classification quality for 8 neighbors and 500 dimensions. The classification quality of the two baselines clearly degrades as many peers have only small training sets, due to the Zipf distribution. This distribution is ubiquitous for content on the Internet [4], and it is therefore important to be able to cope with it.

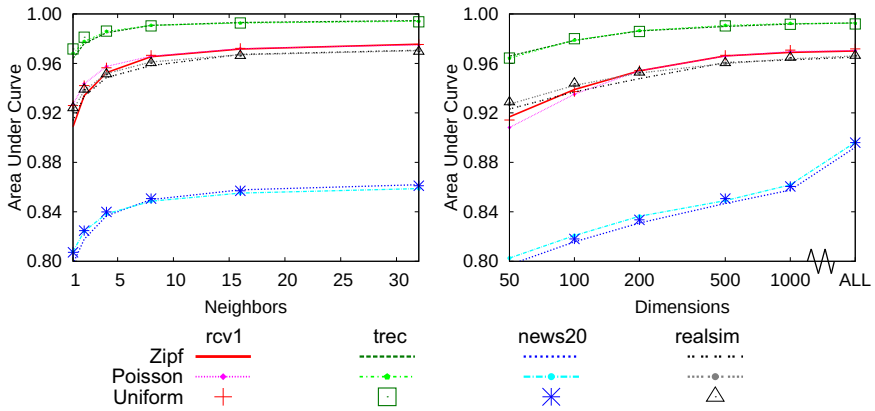

Figure 6.7: Influence of training set distributions: (a) number of neighbors, and, (b) number of dimensions.

Training set		Quality (AUC)			
Dataset	Distr.	SVM	RSVM	CSVM	CRSVM
rcv1	zipf	0.879	0.669	0.965	0.884
	poisson	0.911	0.706	0.965	0.897
	uniform	0.907	0.730	0.966	0.897
trec	zipf	0.932	0.677	0.990	0.941
	poisson	0.956	0.738	0.990	0.945
	uniform	0.959	0.813	0.990	0.946
news20	zipf	0.793	0.608	0.849	0.733
	poisson	0.816	0.622	0.848	0.745
	uniform	0.817	0.650	0.850	0.734
realsim	zipf	0.880	0.675	0.958	0.886
	poisson	0.901	0.718	0.961	0.887
	uniform	0.907	0.721	0.960	0.903

Table 6.1: Classification quality for different training set distributions.

6.3.6 Comparison with other Algorithms

In this section, we present the results of our comparison with the state of the art in distributed and collaborative classification. We compared CSVM and CRSVM with two other collaborative classification algorithms, CASC, the state of the art in distributed classification, and HASH/RHASH, which perform dimensionality reduction based on hashing. CENTR was also included in the comparison, as an indication of the maximum possible quality for the given training data.

We compared all algorithms with respect to their cost/quality ratio, i.e., which quality can be achieved with a certain network cost budget. The CSVM and CRSVM algorithms were initialized with 8 neighbors per peer and d dimensions per model, where $50 \leq d \leq ALL$. As the number of neighbors was fixed, for each network budget the affordable number of dimensions was precomputed according to our cost model (see Section 6.2). HASH and RHASH were configured to yield the same network cost as the CSVM/CRSVM algorithms. This involved setting the number of dimensions (the HASH buckets) to d, and the neighborhood size to 16 per peer (because HASH has

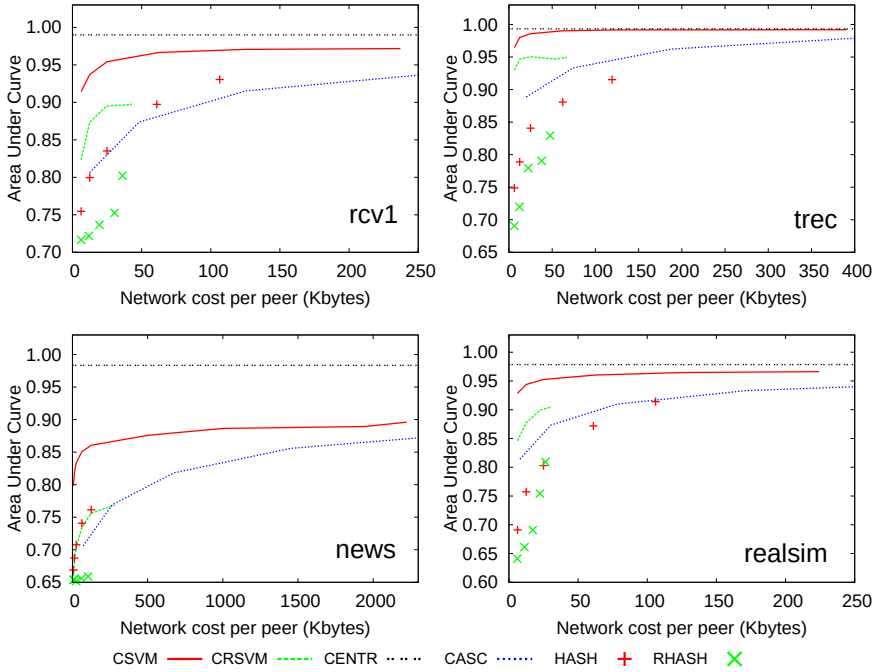

Figure 6.8: Classification quality for different algorithms

half the transmission cost per dimension compared to CSVM). For CASC, we tested all possible configurations and chose the ones performing best for a given network budget. Note that CASC does not allow preselecting or upper-bounding the network cost, and therefore the cost ranges of the compared algorithms do not completely overlap. For example, for the rcv1 collection, there exists no possible configuration of CASC with a network cost less than 10 Kbytes per peer.

Figures 6.8(a)-(d) depict the AUC measures in correlation to the network requirements for all compared algorithms. We see that CSVM substantially outperforms all other distributed algorithms. For three of the datasets, it closely approximates the quality of CENTR, with very small network cost. The only exception is the news20 dataset, which is challenging for all distributed algorithms. This can be explained by the characteristics of the news20 classification task: the classifier needs to identify messages from 10 newsgroups as positive, and from 10 other ones as negative. As such, both the positive and negative class consist of a high variety of topics (e.g., 'cars', 'sports', 'politics', 'religion'). The available local training documents are not sufficient to capture this variety, leading to local classifiers of low quality. Nonetheless, CSVM helps increasing the quality, and achieves the highest AUC of the distributed approaches.

CRSVM is inferior to CSVM, but still outperforms the HASH and CASC algorithms in its cost range. We expect RSVM to show its strengths only with very large local training sets. Another limitation of CRSVM and of all the RSVM-based algorithms also becomes apparent from these results: the underlying RSVM already trades classifier accuracy for efficiency, limiting the possibility of a fine-grained control of the desired cost/quality tradeoff.

It is also interesting to consider the point where CASC reaches the maximum qual-

Dataset	Quality (AUC)	Network cost per peer (Kbytes)	
		CSVM	CASC
rcv1	0.971	237	2342
trec	0.992	390	1400
news20	0.895	2223	5157
realsim	0.966	224	2006

Table 6.2: Network cost of CSVM and CASC for achieving a comparable classification quality.

ity of the considered CSVM configuration (for improving the CSVM quality further, more neighbors would be required). Table 6.2 presents the network cost of the algorithms. We observe that CASC incurs up to an order of magnitude higher cost than CSVM for achieving the same classification quality. The other compared algorithms are not depicted in the table as they cannot achieve a quality level comparable to CSVM.

6.4 Summary

We have presented CSVM, a collaborative classification algorithm built on top of a P2P network. Each participating node merges SVM classifiers trained locally on a small number of neighboring nodes into more accurate meta classifiers. To reduce the network cost, only the most important components of the SVM models are exchanged between users.

Our experimental evaluation provided a systematic study of the system parameters, i.e., number of collaborating neighbors and dimensionality of the models. The results demonstrate that CSVM substantially outperforms state of the art collaborative classification techniques while keeping network costs an order of magnitude lower. Our approach offers the additional advantage that network load can be controlled in a precise and flexible way, allowing for an optimal utilization of network resources.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, we have addressed problems in the area of P2P information retrieval and data mining, proposing novel and efficient algorithms that advance the state of the art.

In particular, in the context of information retrieval we have focused mainly on two frequently occurring problems, keyword search and near duplicate detection. With respect to keyword search, we showed that efficiency of the state of the art P2P systems can be drastically increased by optimizing the maintenance of the distributed inverted index, without sacrificing effectiveness. The key innovation of our optimization lies in the deployment of a middle layer between the peers and the distributed inverted index, for coordinating the indexing process. PCIR forms small groups of peers around super peers, which undertake the responsibility of the DHT maintenance process for the whole group. We explored two different peer grouping methods, and showed that additional performance improvement can be observed by clustering the peers to increase the term overlap in super peers. Our experimental results show an order of magnitude improvement compared to the state of the art approaches for constructing indexes of various granularities, and without overloading any of the peers. The accompanying theoretical analysis confirms the applicability of the experimental observations for different setups.

With respect to near duplicate detection, we demonstrated experimentally the importance of fine tuning the core parameters of LSH-based near duplicate detection methods for optimizing the network cost. In sharp contrast to existing methods, our proposal determines the optimal values of these parameters dynamically, by exploiting statistics which can be inexpensively collected from the network. POND relies on a thorough theoretical analysis for linking the network cost with probabilistic guarantees, and deriving the optimal values of the number of LSH hash tables and hash functions. The approach is also the first one addressing the video linkage problem over a P2P environment. A large-scale experimental evaluation with 227 Gbytes of real-world text and multimedia datasets confirmed that POND satisfies the desired probabilistic guarantees with the optimal network cost, and showed improvements of several orders of magnitude compared to the case where the parameters are not determined dynamically.

Concerning data mining, we have considered P2P clustering and classification, mainly focusing on textual data. In the context of clustering, we have proposed PCP2P, which exploits the Zipf distribution of term frequencies inside each document to effi-

ciently detect the relevant clusters over a DHT network in two stages. We have explored three different strategies for excluding the irrelevant clusters, and derived probabilistic correctness guarantees. In addition to the theoretical evaluation, the good performance of all PCP2P variants was demonstrated with large-scale experiments, using several text corpora, and by comparing with the state of the art algorithms for P2P clustering.

Finally, we have demonstrated the benefits of collaboration for increasing the classification accuracy, and proposed a framework which enables collaborative classification over unstructured P2P networks. The novel aspect of our framework is that it combines meta classifiers with dimensionality reduction, for increasing the classification accuracy and decreasing the network cost for collaboration. We have considered two instantiations of the framework, the one using the state of the art SVM classifiers, and the second employing Reduced SVMs, which trade quality for efficiency. Our experiments with four standard datasets established the good properties of the proposed framework, and its superior efficiency and effectiveness with the state of the art competitive algorithms. We have also studied the involved cost/quality tradeoffs, and showed that a near-optimal performance is achieved with a network cost that is sufficiently small to allow deployment of the framework over mobile networks.

A common aspect of our contributions is that they enable a tradeoff between efficiency and effectiveness based on their capabilities of the participating peers and the application requirements. All algorithms closely approximate the optimal quality of their corresponding centralized implementations, but without requiring dedicated severs, without suffering from bottlenecks, and at a small fraction of the cost of the state of the art counterparts. Having a clear focus on large-scale P2P networks, the algorithms scale well with the number of peers and do not make any assumption about the distribution of documents to peers, or the capacity and uptime of the participating peers.

7.2 Future Work

In addition to the particular extensions of each contribution, which we already described in the previous chapters, our future work focuses on combining PCIR and PCP2P towards constructing a fully distributed advanced information retrieval solution. Even though clustering was already considered in the literature for improving the IR quality, e.g., [101, 176], past solutions assumed a central architecture for performing the clustering, and therefore suffered from scalability limitations. Having proposed a fully distributed and scalable P2P clustering algorithm, we want to pursue this idea further, incorporating the state of the art in cluster-based information retrieval for a fully decentralized P2P IR algorithm.

Another aspect of our work focuses on applying the core ideas of PCP2P to other clustering algorithms, not necessarily distributed, for enabling probabilistic pruning of the candidate clusters. As already demonstrated, this pruning enables substantial reduction of the required document-cluster comparisons, which translate to drastic computational cost reductions. Therefore, it is also interesting to pursue this idea for centralized clustering algorithms, for improving their efficiency without a negative influence on quality.

With respect to POND, we want to repeat the described theoretical analysis for other P2P configurations [14, 68]. These systems currently address NDD as a special case of the KNN problem, and therefore POND cannot be applied directly to optimize the network cost. Specializing the functionality of these systems to only NDD

queries with fixed distance thresholds would enable the application of our optimization method. Finally, we will explore new application scenarios of NDD in multimedia mining and retrieval over existing P2P file sharing networks, such as Limewire. We expect that the proposed technique finds direct applications to metadata-based search, where annotations (such as tags or descriptions) provided by different users for near duplicate content can be combined to build more comprehensive indexes.

Appendix A

Proofs

A.1 Proofs for Chapter 3

We first introduce two Lemmas which are necessary for our proofs.

Lemma A.1. *The expected number of true bits in a Bloom filter of length m with k hash functions after n elements were hashed is: $\hat{S}(n) = m\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)$.*

Proof. Given a Bloom filter of length m with k hash functions and n elements hashed into it. We define the binary random variables $Z_1, Z_2, \ldots, Z_m$ where Z_i is interpreted to be the indicator variable for the event that the ith bit in the Bloom filter is set to true. The probability that the ith bit is set to true is $P[i = true] = 1 - \left(1 - \frac{1}{m}\right)^{kn}$. Having a Bloom filter of length m, the expected number of true bits equals to $\hat{S}(n) = \sum_{i=1}^{m} P[i = true] = m\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)$. □

Lemma A.2. *Given a Bloom filter BF of length m with k hash functions and t bits set to true. The expected number of elements hashed in the Bloom filter is $\hat{S}^{-1}(t) = \frac{\ln\left(1 - \frac{t}{m}\right)}{k \times \ln\left(1 - \frac{1}{m}\right)}$*

Proof. We find $\hat{S}^{-1}(t)$ using the probability of a bit to be true:

$$P[i = true] = \frac{t}{m} = 1 - \left(1 - \frac{1}{m}\right)^{k\hat{S}^{-1}(t)} \Rightarrow \left(1 - \frac{1}{m}\right)^{k\hat{S}^{-1}(t)} = 1 - \frac{t}{m} \Rightarrow$$

$$\hat{S}^{-1}(t) = \frac{\ln\left(1 - \frac{t}{m}\right)}{k \times \ln\left(1 - \frac{1}{m}\right)}$$

□

Theorem 3.1

Proof. We now show how the cardinality of the intersection of two sets A and B can be derived from their Bloom filters. 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. A.1), 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 Lemma A.2 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 we denote the number of random collisions, that is, the 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 $\frac{|SET_{BF_A}|-|SET_{BF_\cap}|}{m-|SET_{BF_\cap}|} \times \frac{|SET_{BF_B}|-|SET_{BF_\cap}|}{m-|SET_{BF_\cap}|}$, 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:

$$\hat{R} = (m - |SET_{BF_\cap}|) \times \frac{|SET_{BF_A}|-|SET_{BF_\cap}|}{m-|SET_{BF_\cap}|} \times \frac{|SET_{BF_B}|-|SET_{BF_\cap}|}{m-|SET_{BF_\cap}|}$$

Moreover, by definition:

$$|SET_{BF_\wedge}| = |SET_{BF_\cap}| + R \tag{A.1}$$

By replacing R in Equation A.1 with the expected value we get an estimation for $|SET_{BF_\cap}|$:

$$E(|SET_{BF_\cap}|) = |SET_{BF_\wedge}| - (m - |SET_{BF_\cap}|) \times \frac{|SET_{BF_A}|-|SET_{BF_\cap}|}{m-|SET_{BF_\cap}|} \times \frac{|SET_{BF_B}|-|SET_{BF_\cap}|}{m-|SET_{BF_\cap}|} \tag{A.2}$$

Note that $BF_\cap$ is a normal Bloom filter of the set $A \cap B$. Thus we can use Lemma A.1 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 A.2, we get an estimation for $E(|A \cap B|)$:

$$\begin{aligned} E(|A \cap B|) = &\frac{\ln(m^2 - m|SET_{BF_A}| - m|SET_{BF_B}| + |SET_{BF_A}| \times |SET_{BF_B}|)}{k \ln(1 - 1/m)} - \\ &\frac{\ln(m^2 - m \times |SET_{BF_A}| - m \times |SET_{BF_B}| + m \times |SET_{BF_\wedge}|)}{k \ln(1 - 1/m)} \end{aligned} \tag{A.3}$$

Equation 3.2 is derived by replacing the notation of $|SET_{BF_x}|$ with the original notation of $tb(BF_x)$.

For completeness, we mention that it is also possible to derive probabilistic upper and lower bounds for the value of $E(|A \cap B|)$, which can be useful for several scenarios. The same applies for the value of $\hat{S}^{-1}(t)$, derived by Lemma A.2. We derive these

bounds in [133]. □

Theorem 3.2

Proof. We use Theorem 3.1 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. For the lower bound, we use the simplified form proposed in [121], pp. 69–70.

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

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

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

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

which gives:

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

□

Theorem 3.3

Proof. 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. Formally, $DC(g) := \bigcup_{p \in g} DC(p)$.

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

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

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

$$E\left(\frac{D_p}{D_g}\right) = \frac{len(DC(p))^\beta}{(n/n_{sp} \times len(DC(p)))^\beta} = \left(n_{sp}/n\right)^\beta \tag{A.4}$$

We now look at the cost ratio between basic PCIR and the flat DHT indexing approach. The term $n_{sp} \times D_g \times \log(n)$ is the dominant term in the equation for the basic PCIR cost expression (Equation 3.5), and closely approximates the total cost. This gives:

$$E\left(\frac{C_{basic}}{C_{flat}}\right) \approx \frac{n_{sp} \times D_g(\log(n) + 1)}{n \times D_p \times (\log(n) + 1)} \tag{A.5}$$

From Equations A.4 and A.5 we get:

$$E\left(\frac{C_{basic}}{C_{flat}}\right) \approx \frac{n_{sp} \times (n/n_{sp})^\beta}{n} = \left(n_{sp}/n\right)^{1-\beta} \tag{A.6}$$

□

A.2 Proofs for Chapter 4

Lemma 4.1

Proof. $Pr\left[Label_j(x) = Label_j(y)\right]$ can be expressed as a product of the individual probabilities of the k bits in the corresponding labels to match. As explained in the previous section, the individual bits of the labels are set using min-wise hashing followed by binary hashing. If the result of min-wise hashing of two resources is the same, the result of the binary hashing is also assured to be the same. If the result of the min-wise hashing is not the same, the result of binary hashing can still be the same with probability 0.5. More specifically, the probability that x and y have the same label $Label_j$ is computed as follows. Consider the min-wise hashing value computed for a single hash function $f_i(\cdot) \in \mathcal{H}_j$. The probability that min-wise hashing of the two resources using hash function $f_i(\cdot)$ yields the same result depends on the similarity of the resources. For the case that $Sim(x, y)$ denotes Jaccard similarity between the two resources, Broder et al. [24] show the following:

$$Pr[\min\{f_i(x)\} = \min\{f_i(y)\}] = Sim(x, y) \tag{A.7}$$

where $\min\{f_i(\cdot)\}$ denotes the min-wise hashing values of the resources using f_i. Because of pairwise independence of the hash functions in $\mathcal{H}_j$, the probability that i of the k min-wise hash values of x and y are pairwise equal is $Sim(x, y)^i$. It follows that the remaining $(k - i)$ min-wise hash values are not pairwise equal with a probability of $(1 - Sim(x, y))^{k-i}$. Owing to binary hashing that follows the min-wise hashing, the probability of these $(k - i)$ min-hash values to still map to equal binary values is given by $((1 - Sim\,(x, y)) \times 0.5)^{k-i}$. Factor $\binom{k}{i}$ accounts for all combinations of i out of k, for $0 \leq i \leq k$. □

Corollary 4.1

Proof. Two corresponding labels of x and y do not match with a probability of $(1 - Pr[Label(x) = Label(y)])$, which can be computed with Lemma 4.1. The probability that none of the labels of x and y match is $(1 - Pr[Label(x) = Label(y)])^l$. The probability that at least one of the l labels matches follows directly: $pr_{\text{found}}(x, y) = 1 - (1 - Pr[Label(x) = Label(y)])^l = 1 - \left(1 - \sum_{i=0}^{k} \binom{k}{i} \times (1 - Sim(x, y))^i \times Sim(x, y)^{k-i} \times 0.5^i\right)^l$. □

Theorem 4.1

Proof. We first show that the probability value $pr_{\text{found}}(x, y)$ monotonically increases with $Sim(x, y)$. The derivative of $pr_{\text{found}}(x, y)$ with respect to $Sim(x, y)$ is

$$pr'_{\text{found}}(x, y) = \frac{k\left(0.5 + \frac{0.5}{Sim(x,y)}\right)^k Sim(x, y)^k}{1 + Sim(x, y)}$$

Notice that pr'_{found} is always positive for $0 \leq Sim(x, y) \leq 1$, which means that $pr_{\text{found}}(x, y)$ is monotonically increasing with $Sim(x, y)$. Hence, for x, y with $Sim(x, y) \geq minSim$,

$$1 \geq pr_{\text{found}}(x, y) \geq 1 - \left(1 - \sum_{i=0}^{k} \binom{k}{i} \times (1 - minSim)^i \times minSim^{k-i} \times 0.5^i\right)^l$$

□

Theorem 4.2

Proof. Probability pr_{ndd} monotonically increases with the decrease of k, if all other parameters are fixed (Eqn. 4.2). Therefore, for all $k \leq k_0$, the value of pr_{ndd} will be higher than or equal to the probability pr_{min} desired by the user. The value of k is orthogonal to maintenance cost, but affects query execution cost due to the false positives. The number of false positives monotonically decreases with k. Therefore, the number of false positives will be reduced by selecting the maximum k value for which $pr_{\text{ndd}} \geq pr_{min}$. This value is $k = \lfloor k_0 \rfloor$. □

A.3 Proofs for Chapter 5

Theorem 5.1

Proof. The proof uses the simplified Chernoff bound for the lower tail ([121], p. 72), which states that for any $\delta > 0$, the probability of the sum of N independent Poisson trials to be less than $(1 - \delta) \times \mu$ is less than $\exp(-\mu \times \delta^2/2)$. Symbol μ denotes the expected value for the sum.

We apply the bounds to find $Pr[TF(t, c_i) < DocTF_{min}]$ as follows. For a document d of length l and for a term t, we define binary random variables $Z_1, \ldots, Z_l$, where Z_i denotes the event that the ith term of d is t. As in most language models, we assume that terms are independent. The expected number of occurrences of t is denoted by

$\hat{TF}(t,d) = \phi_i[t] \times l$. Then, by Chernoff bounds (lower tail):

$$Pr[TF(t,d) \geq DocTF_{min}] \geq$$
$$1 - \exp(-\hat{TF}(t,d) \times (1 - DocTF_{min}/\hat{TF}(t,d))^2/2)$$

For any probability Pr_{min}, the minimum value of $DocTF_{min}$ satisfying $Pr[TF(t,d) \geq DocTF_{min}] \geq Pr_{min}$ is:

$$DocTF_{min} = \hat{TF}(t,d) - \sqrt{2 \times \hat{TF}(t,d) \times \log(\frac{1}{1 - Pr_{min}})} \tag{A.8}$$

Since $DocTF_{min}$ is a natural number, we take the floor of the RHS of expression A.8 as its value. □

Theorem 5.2

Proof. We represent clusters as concatenations of their documents. Since we assume that all documents in cluster c_i follow the same language model ϕ_i, the cluster also follows ϕ_i. The equations follow directly from Chernoff inequality, similar to Theorem 5.1. □

Appendix B

Publications

Journal articles

1. Odysseas Papapetrou,Wolf Siberski, Wolfgang Nejdl. PCIR: Combining DHTs and Peer Clusters for Efficient Full-text P2P Indexing, Computer Networks 54(12): 2019-2040 (2010), Elsevier.

2. Odysseas Papapetrou, Wolf Siberski, Wolfgang Nejdl. Cardinality estimation and dynamic length adaptation for Bloom filters, Distributed and Parallel Databases, 28(2):119-156 (2010), Springer.

3. George A. Papadopoulos, Aristos Stavrou, Odysseas Papapetrou. An implementation framework for Software Architectures based on the coordination paradigm. Science of Computer Programming 60(1): 27-67 (2006), Elsevier.

Peer-reviewed conferences

1. Odysseas Papapetrou, Ekaterini Ioannou, Dimitrios Skoutas. Efficient Discovery of Frequent Subgraph Patterns in Uncertain Graph Databases, in: Proc. 14th International Conference on Extending Database Technology (EDBT), 2011, Uppsala, Sweden.

2. Odysseas Papapetrou, Ling Chen. XStreamCluster: an Efficient Algorithm for Streaming XML data Clustering, in: Proc. 16th Database Systems For Advanced Applications (DASFAA), 2011, Hong Kong.

3. Odysseas Papapetrou, Wolf Siberski, Stefan Siersdorfer. Collaborative Classification over P2P networks, in: Proc. WWW 2011 (Companion Volume), Hyderabad, India.

4. Odysseas Papapetrou, Wolf Siberski, Norbert Fuhr. Text Clustering for Peer-to-Peer Networks with Probabilistic Guarantees, in: Proc. 32nd European Conference on Information Retrieval (ECIR), 2010, Milton Keynes, UK.

5. Odysseas Papapetrou, Sukriti Ramesh, Stefan Siersdorfer, Wolfgang Nejdl. Optimizing Near Duplicate Detection for P2P Networks, in: Proc. IEEE International Conference on Peer-to-Peer Computing (P2P), 2010, Delft, Netherlands.

6. Odysseas Papapetrou, George Papadakis, Ekaterini Ioannou, Dimitrios Skoutas. Efficient Term Cloud Generation for Streaming Web Content, in: Proc. 10th International Conference on Web Engineering (ICWE), 2010, Vienna, Austria.

7. Ekaterini Ioannou, Odysseas Papapetrou, Dimitrios Skoutas, Wolfgang Nejdl. Efficient Semantic-Aware Detection of Near Duplicate Resources, in: Proc. 7th Extended Semantic Web Conference (ESWC), 2010, Heraklion, Greece.

8. Sukriti Ramesh, Odysseas Papapetrou, Wolf Siberski. Optimizing Distributed Joins with Bloom filters, in: Proc. Fifth International Conference on Distributed Computing and Internet Technologies (ICDCIT), 2008, New Delhi, India.

9. Loizos Michael, Wolfgang Nejdl, Odysseas Papapetrou, Wolf Siberski. Improving distributed join efficiency with extended Bloom filter operations, in: Proc. IEEE 21st International Conference on Advanced Information Networking and Applications (AINA), 2007, Niagara Falls, Canada.

10. Odysseas Papapetrou, Wolf Siberski, Wolf-Tilo Balke, Wolfgang Nejdl. DHTs over Peer Clusters for Distributed Information Retrieval, in: Proc. IEEE 21st International Conference on Advanced Information Networking and Applications (AINA), 2007, Niagara Falls, Canada.

11. Juri L. De Coi, Eelco Herder, Arne Koesling, Christoph Lofi, Daniel Olmedilla, Odysseas Papapetrou, and Wolf Siberski. A model for competence gap analysis, in: Proc. International Conference on Web Information Systems and Technology (WEBIST), 2007, Barcelona, Spain.

12. Odysseas Papapetrou, Sebastian Michel, Matthias Bender, Gerhard Weikum. On the Usage of Global Document Occurrences in Peer-to-Peer Information Systems, in: Proc. International Conference of Cooperative Information Systems (CoopIS), 2005, Ag. Napa, Cyprus.

13. Odysseas Papapetrou, George A. Papadopoulos. Aspect Oriented Programming for a component-based real life application: a case study, in: Proc. of Symposium of Applied Computing (SAC), 2004, Nicosia, Cyprus.

14. Odysseas Papapetrou, George Samaras. IPMicra: Toward a Distributed and Adaptable Location Aware Web Crawler, in: Local Proc. Advances in Databases and Information Systems (ADBIS) 2004, Budapest, Hungary.

15. Odysseas Papapetrou, George Samaras. Minimizing the Network Distance in Distributed Web Crawling, in: Proc. International Conference of Cooperative Information Systems (CoopIS), 2004, Larnaca, Cyprus.

16. Odysseas Papapetrou, George Samaras. IPMicra: An IP-address based Location Aware Distributed Web Crawler, in: Proc. International Conference on Internet Computing, 2004, Nevada, USA.

17. Odysseas Papapetrou, George Samaras. Distributed location aware web crawling, in: Proc. WWW (Alternate Track Papers & Posters), 2004, New York, USA.

18. Odysseas Papapetrou, Stavros Papastavrou, George Samaras. UCYMICRA: Distributed Indexing of the Web Using Migrating Crawlers, in Proc. Advances in Databases and Information Systems (ADBIS), 2003, Dresden, Germany.

19. Odysseas Papapetrou, Stavros Papastavrou, George Samaras. Distributed Indexing of the Web using Migrating Crawlers, in: Proc. WWW (Alternate Track Papers & Posters), 2003, Budapest, Hungary.

Peer-reviewed workshops

1. Odysseas Papapetrou, Wolf Siberski, Fabian Leitritz, Wolfgang Nejdl. Exploiting Distribution Skew for Scalable P2P Text Clustering Databases, in: Proc. Information Systems and Peer-to-Peer Computing (DBISP2P) 2008, Auckland, New Zealand.

2. Odysseas Papapetrou. Full-text Indexing and Information Retrieval in P2P systems, in: Proc. 11th International Conference on Extending Database Technology, PhD Workshop (EDBT), 2008, Nantes, France.

Technical reports

1. Odysseas Papapetrou, Wolf Siberski, Norbert Fuhr. Decentralized Probabilistic Text Clustering, under revision at TKDE, 2010.

2. Odysseas Papapetrou, Wolf Siberski, Stefan Siersdorfer. Efficient Model Sharing for Collaborative Text Classification, 2011.

Bibliography

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

[2] K. Aberer, P. Cudré-Mauroux, M. Hauswirth, and T. V. Pelt. GridVine: Building internet-scale semantic overlay networks. In *ISWC*, pages 107–121, 2004.

[3] K. Aberer, F. Klemm, M. Rajman, and J. Wu. An architecture for peer-to-peer information retrieval. In *Workshop on Peer-to-Peer Information Retrieval*, 2004.

[4] L. A. Adamic and B. A. Huberman. Zipf's law and the internet. *Glottometrics*, 3:143–150, 2002.

[5] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[6] H. H. Ang, V. Gopalkrishnan, S. C. H. Hoi, and W. K. Ng. Cascade rsvm in peer-to-peer networks. In *ECML/PKDD (1)*, pages 55–70, 2008.

[7] H. H. Ang, V. Gopalkrishnan, W. K. Ng, and S. C. H. Hoi. Communication-efficient classification in P2P networks. In *ECML/PKDD (1)*, pages 83–98, 2009.

[8] J. Attenberg, K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and M. Zinkevich. Collaborative email-spam filtering with consistently bad labels using feature hashing. In *CEAS*, 2009.

[9] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. DL meets P2P - distributed document retrieval based on classification and content. In *ECDL*, pages 379–390, 2005.

[10] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive distributed top k retrieval in peer-to-peer networks. In *ICDE*, pages 174–185, 2005.

[11] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu, and S. Datta. Clustering distributed data streams in peer-to-peer environments. *Inf. Sci.*, 176(14):1952–1985, 2006.

[12] D. Barbalace, C. Lucchese, C. Mastroianni, S. Orlando, and D. Talia. Mining@home : Public resource computing for distributed data mining. In *From Grids to Service and Pervasive Computing*. Springer, 2008.

[13] C. Batten, K. Barr, A. Saraf, and S. Trepetin. pStore: A secure peer-to-peer backup system. Technical Memo MIT-LCS-TM-632, Massachusetts Institute of Technology Laboratory for Computer Science, October 2002.

[14] M. Bawa, T. Condie, and P. Ganesan. LSH forest: self-tuning indexes for similarity search. In *WWW*, pages 651–660, 2005.

[15] M. Bender, S. Michel, P. Triantafillou, and G. Weikum. Global document frequency estimation in peer-to-peer web search. In *WebDB*, 2006.

[16] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Improving collection selection with overlap awareness in P2P search engines. In *SIGIR*, pages 67–74, 2005.

[17] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Minerva: Collaborative P2P search. In *VLDB*, pages 1263–1266, 2005.

[18] P. N. Bennett, S. T. Dumais, and E. Horvitz. Probabilistic combination of text classifiers using reliability indicators: models and results. In *SIGIR*, pages 207–214, 2002.

[19] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In *SIGCOMM*, pages 353–366, 2004.

[20] C. Blake. A comparison of document, sentence, and term event spaces. In *ACL*, 2006.

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

[22] W. J. Bolosky, J. R. Douceur, and J. Howell. The Farsite project: a retrospective. *SIGOPS Oper. Syst. Rev.*, 41:17–26, April 2007.

[23] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[24] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations (extended abstract). In *STOC*, pages 327–336, 1998.

[25] C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, 1998.

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

[27] M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.

[28] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P systems scalable. In *SIGCOMM*, pages 407–418, 2003.

[29] L. Chen, P. Wright, and W. Nejdl. Improving music genre classification using collaborative tagging data. In *WSDM*, pages 84–93, 2009.

[30] I. Clarke, S. G. Miller, T. W. Hong, O. Sandberg, and B. Wiley. Protecting Free Expression Online with Freenet. *IEEE Internet Computing*, 6:40–49, 2002.

[31] B. F. Cooper. Guiding queries to information sources with InfoBeacons. In *Middleware*, pages 59–78, 2004.

[32] G. V. Cormack. TREC 2007 Spam Track Overview. In *Text REtrieval Conference, (TREC)*, 2007.

[33] G. V. Cormack and T. R. Lynam. Online supervised spam filter evaluation. *ACM Trans. Inf. Syst.*, 25(3):11, 2007.

[34] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS using chord. In *IPTPS*, Cambridge, MA, March 2002.

[35] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *ICDCS*, pages 23–, 2002.

[36] A. Crespo and H. Garcia-Molina. Semantic overlay networks for P2P systems. In *AP2PC*, pages 1–13, 2004.

[37] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In *HPDC*, pages 236–249, 2003.

[38] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris. Designing a DHT for low latency and high throughput. In *NSDI*, pages 85–98, 2004.

[39] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, pages 253–262, 2004.

[40] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta. Distributed data mining in P2P networks. *IEEE Internet Computing*, 10(4):18–26, 2006.

[41] S. Datta, C. Giannella, and H. Kargupta. K-Means clustering over a large, dynamic network. In *Proc. SDM*, 2006.

[42] S. Datta, C. R. Giannella, and H. Kargupta. Approximate distributed K-Means clustering over a P2P network. *TKDE*, 21(10):1372–1388, 2009.

[43] I. S. Dhillon and D. S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Proceedings of the Workshop on Large-Scale Parallel KDD Systems*, pages 245–260, San Diego, CA, USA, 1999.

[44] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li. Modeling LSH for performance tuning. In *CIKM*, pages 669–678. ACM, 2008.

[45] C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis. DESENT: decentralized and distributed semantic overlay generation in P2P networks. *IEEE Journal on Selected Areas in Communications*, 25(1):25–34, 2007.

[46] S. T. Dumais, J. C. Platt, D. Hecherman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *CIKM*, 1998.

[47] M. Eisenhardt, W. Müller, and A. Henrich. Classifying documents by distributed P2P clustering. In *INFORMATIK*, pages 286–291, Frankfurt, Germany, 2003.

[48] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured P2P networks. In *IPTPS*, pages 304–314, 2003.

[49] S. Ertekin, J. Huang, L. Bottou, and L. Giles. Learning on the border: active learning in imbalanced data classification. In *CIKM*, pages 127–136, 2007.

[50] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[51] F. Falchi, C. Gennaro, F. Rabitti, and P. Zezula. A distributed incremental nearest neighbor algorithm. In *INFOSCALE*, page 82, 2007.

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

[53] T. Fawcett. An introduction to ROC analysis. *Pattern Recogn. Lett.*, 27(8):861–874, 2006.

[54] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7, 1936.

[55] G. Forman. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.*, 3:1289–1305, 2003.

[56] G. Forman and B. Zhang. Distributed data clustering can be efficient and exact. *SIGKDD Explor. Newsl.*, 2(2):34–38, 2000.

[57] Y. Freund. An Adaptive Version of the Boost by Majority Algorithm. In *COLT*, 1999.

[58] J. Fürnkranz, E. Hüllermeier, E. Loza Mencía, and K. Brinker. Multilabel classification via calibrated label ranking. *Mach. Learn.*, 73(2):133–153, 2008.

[59] A. Ganapathiraju, J. Hamaker, and J. Picone. Support vector machines for speech recognition. In *International Conference on Spoken Language Processing*, pages 2348–2355, 1998.

[60] P. Ganesan, P. K. Gummadi, and H. Garcia-Molina. Canon in G major: Designing DHTs with hierarchical structure. In *ICDCS*, pages 263–272, 2004.

[61] P. Ganesan, Q. Sun, and H. Garcia-Molina. Adlib: A self-tuning index for dynamic P2P systems. In *ICDE*, pages 256–257, 2005.

[62] A. J. Ganesh, A.-M. Kermarrec, E. L. Merrer, and L. Massoulié. Peer counting and sampling in overlay networks based on random walks. *Distributed Computing*, 20(4):267–278, 2007.

[63] J. Gao and P. Steenkiste. An adaptive protocol for efficient support of range queries in DHT-based systems. In *ICNP*, pages 239 – 250, 2004.

[64] P. Garbacki, D. H. J. Epema, and M. van Steen. Optimizing peer relationships in a super-peer network. In *ICDCS*, page 31, 2007.

[65] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[66] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *INFOCOM*, pages 241–263, 2004.

[67] B. Godfrey, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica. Load balancing in dynamic structured P2P systems. In *INFOCOM*, 2004.

[68] P. Haghani, S. Michel, and K. Aberer. Distributed similarity search in high dimensions using locality sensitive hashing. In *EDBT*, pages 744–755, 2009.

[69] K. M. Hammouda and M. S. Kamel. HP2PC: Scalable hierarchically-distributed peer-to-peer clustering. In *SDM*, 2007.

[70] B. Haslhofer and P. Knezević. The BRICKS digital library infrastructure. In *Semantic Digital Libraries*, pages 151–161. 2009.

[71] H. S. Heaps. *Information Retrieval-Computational and Theoretical Aspects*. Academic Press, 1978.

[72] P. Heymann, G. Koutrika, and H. Garcia-Molina. Can social bookmarking improve web search? In *WSDM*, pages 195–206, 2008.

[73] H.-C. Hsiao and C.-T. King. Similarity discovery in structured P2P overlays. In *ICPP*, pages 636–, 2003.

[74] R. Huebsch, B. N. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi. The architecture of PIER: an internet-scale query processor. In *CIDR*, pages 28–43, 2005.

[75] Y. Ioannidis, D. Maier, S. Abiteboul, P. Buneman, S. Davidson, E. Fox, A. Halevy, C. Knoblock, F. Rabitti, H. Schek, and G. Weikum. Digital library information-technology infrastructures. *Int J Digit Libr*, 5(4):266 – 274, 2005.

[76] S. Iyer, A. I. T. Rowstron, and P. Druschel. Squirrel: a decentralized peer-to-peer web cache. In *PODC*, pages 213–222, 2002.

[77] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *WWW*, pages 640–651, 2003.

[78] Y. Ke, R. Sukthankar, L. Huston, Y. Ke, and R. Sukthankar. Efficient near-duplicate detection and sub-image retrieval. In *ACM Multimedia*, pages 869–876, 2004.

[79] H.-s. Kim, J. Lee, H. Liu, and D. Lee. Video linkage: group based copied video detection. In *CIVR*, pages 397–406, 2008.

[80] J. Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC*, pages 163–170, 2000.

[81] M. Kleis, E. K. Lua, and X. Zhou. Hierarchical peer-to-peer networks using lightweight superpeer topologies. In *ISCC*, pages 143–148, 2005.

[82] F. Klemm and K. Aberer. Aggregation of a term vocabulary for P2P-IR: A DHT stress test. In *DBISP2P*, pages 187–194, 2005.

[83] G. Koloniari and E. Pitoura. A recall-based cluster formation game in P2P systems. *PVLDB*, 2(1):455–466, 2009.

[84] J. Kong, B. Rezaei, N. Sarshar, V. Roychowdhury, and P. Boykin. Collaborative spam filtering using e-mail networks. *IEEE Computer*, 39(8):67–73, 2006.

[85] J. Kubiatowicz, D. Bindel, Y. Chen, S. E. Czerwinski, P. R. Eaton, D. Geels, R. Gummadi, S. C. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Y. Zhao. OceanStore: An architecture for global-scale persistent storage. In *ASPLOS*, pages 190–201, 2000.

[86] A. Kumar, J. Xu, and E. W. Zegura. Efficient and scalable query routing for unstructured peer-to-peer networks. In *INFOCOM*, pages 1162–1173, 2005.

[87] L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley, 2004.

[88] M. Landers, H. Zhang, and K.-L. Tan. Peerstore: Better performance by relaxing in peer-to-peer backup. In *P2P*, pages 72–79, 2004.

[89] V. A. Larsen. Combining audio fingerprints. Technical report, Norwegian University of Science and Technology, 2008. Available at http://daim.idi.ntnu.no/.

[90] L. Lee. Measures of distributional similarity. In *ACL*, 1999.

[91] Y.-J. Lee, O. Mangasarian, and W. Wolberg. Breast cancer survival and chemotherapy: A support vector machine analysis. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 55:1–20, 2000.

[92] Y.-J. Lee and O. L. Mangasarian. RSVM: Reduced support vector machines. In *SDM*, pages 55–70, 2001.

[93] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[94] J. Li, B. T. Loo, J. M. Hellerstein, M. F. Kaashoek, D. R. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *IPTPS*, pages 207–215, 2003.

[95] J. Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil. A performance vs. cost framework for evaluating DHT design tradeoffs under churn. In *INFOCOM*, pages 225–236, 2005.

[96] J. Liang, R. Kumar, and K. W. Ross. The FastTrack overlay: A measurement study. *Computer Networks*, 50(6):842 – 858, 2006.

[97] LIBSVM library and data collections, 2010. Available at http://www.csie.ntu.edu.tw/˜cjlin/libsvm/.

[98] A. Linari and G. Weikum. Efficient peer-to-peer semantic overlay networks based on statistical language models. In *P2PIR*, pages 9–16, 2006.

[99] N. Littlestone and M. Warmuth. The Weighted Majority Algorithm. *Information and Computation*, 108(2):212–261, 1994.

[100] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein. The case for a hybrid P2P search infrastructure. In *IPTPS*, pages 141–150, La Jolla, CA, USA, 2004.

[101] J. Lu and J. Callan. Content-based retrieval in hybrid peer-to-peer networks. In *CIKM '03*, pages 199–206, 2003.

[102] P. Luo, H. Xiong, K. Lü, and Z. Shi. Distributed classification in peer-to-peer networks. In *SIGKDD*, pages 968–976, 2007.

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

[104] Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make Gnutella scalable? In *IPTPS*, pages 94–103, March 2002.

[105] J. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. *5th Berkeley Symposium on Math. Statistics and Probability*, pages 281–297, 1967.

[106] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *PODC*, pages 183–192, 2002.

[107] M. Mani, A.-M. Nguyen, and N. Crespi. What's up 2.0: P2P spontaneous social networking. In *IEEE INFOCOM, Poster session*, 2009.

[108] M. Mani, A.-M. Nguyen, and N. Crespi. Scope: A prototype for spontaneous P2P social networking. In *PerCom Workshops*, pages 220–225, 2010.

[109] C. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[110] L. Massoulie, E. L. Merrer, A.-M. Kermarrec, and A. Ganesh. Peer counting and sampling in overlay networks: random walk methods. In *PODC*, pages 123–132, 2006.

[111] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *IPTPS*, pages 53–65, 2002.

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

[113] M. Mehyar, D. Spanos, J. Pongsajapan, S. H. Low, and R. M. Murray. Asynchronous distributed averaging on communication networks. *IEEE/ACM Trans. Netw.*, 15(3):512–520, 2007.

[114] S. Meregu, S. Srinivasan, and E. Zegura. Adding structure to unstructured peer-to-peer networks: the use of small-world graphs. *Journal of Parallel and Distributed Computing*, 65:142–153, 2005.

[115] S. Merugu and J. Ghosh. Privacy-Preserving Distributed Clustering using Generative Models. In *ICDM*, pages 211–218, 2003.

[116] S. Michel. Top-k aggegation queries in large-scale distributed systems. In *BTW*, pages 418–427, 2009.

[117] S. Michel, M. Bender, N. Ntarmos, P. Triantafillou, G. Weikum, and C. Zimmer. Discovering and exploiting keyword and attribute-value co-occurrences to improve P2P routing indices. In *CIKM*, pages 172–181, 2006.

[118] S. Michel, M. Bender, P. Triantafillou, and G. Weikum. IQN routing: Integrating quality and novelty in P2P querying and ranking. In *EDBT*, pages 149–166, 2006.

[119] E. Minack, R. Paiu, S. Costache, G. Demartini, J. Gaugaz, E. Ioannou, P.-A. Chirita, and W. Nejdl. Leveraging personal metadata for desktop search: The Beagle^{++} system. *J. Web Sem.*, 8(1):37–54, 2010.

[120] D. Mladenić, J. Brank, M. Grobelnik, and N. Milic-Frayling. Feature selection using linear classifier weights: interaction with classification models. In *SIGIR*, pages 234–241, 2004.

[121] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[122] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. T. Schlosser, I. Brunkhorst, and A. Löser. Super-peer-based routing strategies for rdf-based peer-to-peer networks. *J. Web Sem.*, 1(2):177–186, 2004.

[123] T. Neumann, M. Bender, S. Michel, R. Schenkel, P. Triantafillou, and G. Weikum. Distributed top-k aggregation queries at large. *Distributed and Parallel Databases*, 26(1):3–27, 2009.

[124] R. Neumayer, C. Doulkeridis, and K. Nørvåg. Aggregation of document frequencies in unstructured P2P networks. In *WISE*, pages 29–42, 2009.

[125] W. S. Ng, B. C. Ooi, and K.-L. Tan. BestPeer: a self-configurable peer-to-peer system. In *ICDE*, page 272, 2002.

[126] L. T. Nguyen, W. G. Yee, and O. Frieder. Adaptive distributed indexing for structured peer-to-peer networks. In *CIKM*, pages 1241–1250, 2008.

[127] H. Nottelmann and N. Fuhr. Evaluating different methods of estimating retrieval quality for resource selection. In *SIGIR*, pages 290–297, 2003.

[128] G. Oster, P. Molli, S. Dumitriu, and R. Mondéjar. Uniwiki: A collaborative P2P system for distributed wiki applications. In *WETICE*, pages 87–92, 2009.

[129] S. Overell, B. Sigurbjörnsson, and R. van Zwol. Classifying tags using open content resources. In *WSDM*, pages 64–73, 2009.

[130] C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent semantic indexing: a probabilistic analysis. In *PODS*, pages 159–168, 1998.

[131] O. Papapetrou, S. Michel, M. Bender, and G. Weikum. On the usage of global document occurrences in peer-to-peer information systems. In *CoopIS*, pages 310–328, 2005.

[132] O. Papapetrou, W. Siberski, F. Leitritz, and W. Nejdl. Exploiting distribution skew for scalable P2P text clustering. In *DBISP2P*, pages 1–12, 2008.

[133] O. Papapetrou, W. Siberski, and W. Nejdl. Cardinality estimation and dynamic length adaptation for bloom filters. *Distributed and Parallel Databases*, 28:119–156, 2010. 10.1007/s10619-010-7067-2.

[134] T. Pitoura, N. Ntarmos, and P. Triantafillou. Replication, load balancing and efficient range query processing in dhts. In *EDBT*, pages 131–148, 2006.

[135] T. Pitoura, N. Ntarmos, and P. Triantafillou. Saturn: Range queries, load balancing and fault tolerance in DHT data systems. *TKDE*, 2010.

[136] G. Pitsilis, P. Periorellis, and L. Marshall. A policy for electing super-nodes in unstructured P2P networks. In *AP2PC*, pages 54–61, 2004.

[137] I. Podnar, M. Rajman, T. Luu, F. Klemm, and K. Aberer. Scalable peer-to-peer web retrieval with highly discriminative keys. In *ICDE*, pages 1096–1105, 2007.

[138] B. Qiao, G. Wang, and K. Xie. A self-organized semantic clustering approach for super-peer networks. In *WISE*, pages 448–453, 2006.

[139] S. Ramesh, O. Papapetrou, and W. Siberski. Optimizing distributed joins with bloom filters. In *ICDCIT*, pages 145–156, 2008.

[140] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *SIGCOMM*, pages 161–172, 2001.

[141] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Middleware*, pages 21–40, 2003.

[142] S. C. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *USENIX Annual Technical Conference, General Track*, pages 127–140, 2004.

[143] J. Ritter. Why Gnutella can't scale. no, really. Technical report, Darkridge, Inc., 2001. http://www.darkridge.com/~jpr5/doc/gnutella.html.

[144] R. Rodrigues and P. Druschel. Peer-to-peer systems. *Commun. ACM*, 53(10):72–82, 2010.

[145] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350, 2001.

[146] S. Saroiu, K. P. Gummadi, and S. D. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia Syst.*, 9:170–184, 2003.

[147] H. Schulze and K. Mochalski. Internet study 2007. Technical report, IPOQUE, 2007.

[148] A. Schuster, R. Wolff, and D. Trock. A high-performance distributed algorithm for mining association rules. *Knowl. Inf. Syst.*, 7(4):458–475, 2005.

[149] D. Sculley and G. Wachman. Relaxed online svms for spam filtering. In *SIGIR*, pages 415–422, 2007.

[150] C. Shah. Tubekit: a query-based youtube crawling toolkit. In *JCDL*, page 433, 2008.

[151] R. Siebes. pNear: combining content clustering and distributed hash tables. In *P2PKM*, 2005.

[152] S. Siersdorfer and S. Sizov. Meta methods for model sharing in personal information systems. *ACM Trans. Inf. Syst.*, 26(4):1–35, 2008.

[153] T. Silerston and O. Fourmaux. Measuring P2P iptv systems. In *NOSSDAV*, 2007.

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

[155] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, 2000.

[156] R. Steinmetz and K. Wehrle. *Peer-to-Peer Systems and Applications*. Springer, 2005.

[157] M. Steyvers and T. Griffiths. *Handbook of Latent Semantic Analysis*, chapter Probabilistic Topic Models, pages 427–448. Lawrence Erlbaum, 2007.

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

[159] C. Tang, S. Dwarkadas, and Z. Xu. On scaling latent semantic indexing for large peer-to-peer systems. In *SIGIR*, pages 112–121, 2004.

[160] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information retrieval in structured overlays. *SIGCOMM Comput. Commun. Rev.*, 33(1):89–94, 2003.

[161] D. Tao, X. Tang, X. Li, and X. Wu. Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(7):1088–1099, 2006.

[162] I. J. Taylor. Gnutella. In *From P2P to Web Services and Grids*, Computer Communications and Networks, pages 101–116. Springer London, 2005.

[163] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann. Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search. In *SIGCOMM*, pages 49–60, 2007.

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

[165] J. Vaidya and C. Clifton. Privacy preserving naive bayes classifier for vertically partitioned data. In *SDM*, 2004.

[166] M. Vazirgiannis, K. Nørvåg, and C. Doulkeridis. Peer-to-peer clustering for semantic overlay network generation. In *PRIS*, 2006.

[167] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In *ICNP*, pages 2–11, 2006.

[168] H. F. Witschel. Global term weights in distributed environments. *Information Processing and Management*, 44(3):1049–1061, 2008.

[169] D. Wolpert. Stacked Generalization. *Neural Networks*, 5(2):241–259, 1992.

[170] H. Wu, M. Zubair, and K. Maly. Collaborative classification of growing collections with evolving facets. In *Hypertext*, pages 167–170, 2007.

[171] X. Wu, A. G. Hauptmann, and C.-W. Ngo. Practical elimination of near-duplicates from web video search. In *MULTIMEDIA*, pages 218–227, 2007.

[172] X. Wu, C.-W. Ngo, and Q. Li. Threading and autodocumenting news videos: a promising solution to rapidly browse news topics. *Signal Processing Magazine, IEEE*, 23(2):59–68, March 2006.

[173] C. Xiao, W. W. 0011, X. Lin, and J. X. Yu. Efficient similarity joins for near duplicate detection. In *WWW*, pages 131–140, 2008.

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

[175] Y. Xing, Z. Li, and Y. Dai. Peerdedupe: Insights into the peer-assisted sampling deduplication. In *P2P*, pages 1–10, 2010.

[176] J. Xu and W. B. Croft. Cluster-based language models for distributed retrieval. In *SIGIR*, pages 254–261, 1999.

[177] Y. Yan, B. C. Ooi, and A. Zhou. Continuous content-based copy detection over streaming videos. In *ICDE*, pages 853–862, 2008.

[178] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *ICDCS*, pages 5–14, 2002.

[179] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *ICDE*, pages 49–, 2003.

[180] C. Yang. Peer-to-peer architecture for content-based music retrieval on acoustic data. In *WWW*, pages 376–383, 2003.

[181] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *ICML*, pages 412–420, 1997.

[182] H. Yu, K. C.-C. Chang, and J. Han. Heterogeneous learner for web page classification. In *ICDM*, pages 538–545, 2002.

[183] H. Zhang, A. Goel, and R. Govindan. Using the small-world model to improve freenet performance. *Computer Networks*, 46(4):555–574, 2004.

[184] H. Zhang, A. Kankanhalli, and S. W. Smoliar. Automatic partitioning of full-motion video. *Multimedia Systems*, 1(1):10–28, 1993.

[185] Y. Zhao and G. Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55(3):311–331, 2004.

[186] C. Zheng, G. Shen, S. Li, and S. Shenker. Distributed segment tree: Support of range query and cover query over DHT. In *IPTPS*, 2006.

[187] X. Zhou, L. Chen, A. Bouguettaya, N. Xiao, and J. A. Taylor. An efficient near-duplicate video shot detection method using shot-based interest points. *IEEE Transactions on Multimedia*, 11(5):879–891, 2009.

[188] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *FAST*, pages 1–14, 2008.

[189] Y. Zhu and Y. Hu. Efficient, proximity-aware load balancing for DHT-based P2P systems. *IEEE Trans. Parallel Distrib. Syst.*, 16:349–361, 2005.

[190] G. K. Zipf. *Human Behavior and the Principle of Least-Effort*. Addison-Wesley, Cambridge, MA, 1949.

[191] S. Zöls, Z. Despotovic, and W. Kellerer. Cost-based analysis of hierarchical DHT design. In *Peer-to-Peer Computing*, pages 233–239, 2006.