Peer-to-Peer Techniques for Web Information Retrieval and Filtering

Peer-to-Peer Techniques for Web Information Retrieval and Filtering

Christos Tryfonopoulos

Ph.D. Thesis, Department of Electronic and Computer Engineering

Technical University of Crete, July 2006

Peer-to-Peer Techniques for Web Information Retrieval and Filtering

Christos Tryfonopoulos

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in
Electronic and Computer Engineering

Doctoral Committee

Professor Stavros Christodoulakis, Thesis Supervisor
Associate Professor Manolis Koubarakis, Thesis Supervisor
Assistant Professor Vasilis Samoladas, Member

Technical University of Crete
2006

Acknowledgements

I am greatly indebted to my supervisor, Manolis Koubarakis for his belief in me and his unending support throughout the years of my PhD. His enthusiasm and desire for perfection was an inspiration for me. I will always consider him a friend and a valuable source of advice.

I would also like to thank the members of my doctoral committee, Stavros Christodoulakis, Vasilis Samoladas, Peter Triantafillou, Evaggelia Pitoura, Timos Sellis and Euripides G.M. Petrakis for their comments and suggestions on improvements and extensions of this work.

Special thanks go the rest of my colleagues in the group, Stratos, Erietta and Matoula for being good workmates, but most importantly close friends. Stratos has proofread many of my papers, provided insightful comments and suggestions to improve this work and was more than somebody could ask for a colleague.

Words cannot always express the feelings for special persons that make your life meaningfull. Persons that are a stand in difficult personal moments and a relief after long working hours and work-spoiled weekends. Paraskevi, a heartfull thanks for your endless love and support to all my decisions.

Special thanks go to my family for their love and encouragement throughout all my life endeavors. Their courage in life and their strength in difficulties was an extra motivation for this effort.

The good times spent with Costas, Nikos, Sotiris, Amalia, Stamatis, Georgia and Petros made this thesis an easier task. A big thanks for your indispensable friendship and for making every occasion spent with you a special one.

Throughout this thesis I received financial support from the Technical University

of Crete, from the Greek Ministry of Education through the Heraclitus fellowship program, and from European Commission FP6/IST/FET programme Evergrow. I would like to thank all the people in these institutions for these research grants.

To my father, for his uneven struggle...

Abstract

Much information of interest to humans is today available on the Web. People can easily gain access to information but at the same time, they have to cope with the problem of information overload. Consequently, they have to rely on specialised tools and systems designed for searching, querying and retrieving information from the Web. Currently, Web search is controlled by a few search engines that are assigned the burden to follow this information explosion by utilising centralised search infrastructures. Additionally, users are striving to stay informed by sifting through enormous amounts of new information, and by relying on tools and techniques that are not able to capture the dynamic nature of the Web. In this setting, peer-to-peer Web search seems an ideal candidate that can offer adaptivity to high dynamics, scalability, resilience to failures and leverage the functionality of the traditional search engine to offer new features and services.

In this thesis, we study the problem of peer-to-peer resource sharing in wide-area networks such as the Internet and the Web. In the architecture that we envision, each peer owns resources which it is willing to share: documents, web pages or files that are appropriately annotated and queried using constructs from information retrieval models. There are two kinds of basic functionality that we expect this architecture to offer: information retrieval and information filtering (also known as publish/subscribe or information dissemination). The main focus of our work is on providing models and languages for expressing publications, queries and subscriptions, protocols that regulate peer interactions in this distributed environment and indexing mechanisms that are utilized locally by each one of the peers.

Initially, we present three progressively more expressive data models, $\mathcal{WP}$, $\mathcal{AWP}$

and $\mathcal{AWPS}$, that are based on information retrieval concepts and their respective query languages. Then, we study the complexity of query satisfiability and entailment for models $\mathcal{WP}$ and $\mathcal{AWP}$ using techniques from propositional logic and computational complexity.

Subsequently, we propose a peer-to-peer architecture designed to support full-fledged information retrieval and filtering functionality in a single unifying framework. In the context of this architecture, we focus on the problem of information filtering using the model $\mathcal{AWPS}$, and present centralised and distributed algorithms for efficient, adaptive information filtering in a peer-to-peer environment. We use two levels of indexing to store queries submitted by users.

The first level corresponds to the partitioning of the global query index to different peers using a distributed hash table as the underlying routing infrastructure. Each node is responsible for a fraction of the submitted user queries through a mapping of attribute values to peer identifiers. The distributed hash table infrastructure is used to define the mapping scheme and also manages the routing of messages between different nodes. Our set of protocols, collectively called DHTrie, extend the basic functionality of the distributed hash table to offer filtering functionality in a dynamic peer-to-peer environment. Additionally, the use of a self-maintainable routing table allows efficient communication between the peers, offering significantly lower network load and latency. This extra routing table uses only local information collected by each peer to speed up the retrieval and filtering process.

The second level of our indexing mechanism is managed locally by each peer, and is used for indexing the user queries the peer is responsible for. In this level of the index, each peer is able to store large numbers of user queries and match them against incoming documents. We have proposed data structures and local indexing algorithms that enable us to solve the filtering problem efficiently for large databases of queries. The main idea behind these algorithms is to store sets of words compactly by exploiting their common elements using trie-like data structures. Since these algorithms use heuristics to cluster user queries, we also consider the periodic re-organisation of the query database when the clustering of queries deteriorates.

Our experimental results show the scalability and efficiency of the proposed al-

gorithms in a dynamic setting. The distributed protocols manage to provide exact query answering functionality (precision and recall are the same as those of a centralised system) at a low network cost and low latency. Additionally, the local algorithms we have proposed outperform solutions in the current literature. Our trie-based query indexing algorithms proved more than 20% faster than their counterparts, offering sophisticated clustering of user queries and mechanisms for the adaptive reorganisation of the query database when filtering performance drops.

Contents

List of Tables

List of Figures

List of Abbreviations

P2P	Peer-to-Peer
DHT	Distributed Hash Table
$\mathcal{WP}$	Word Pattern
$\mathcal{AWP}$	Attribute Word Pattern
$\mathcal{AWPS}$	Attribute Word Pattern with Similarity
VSM	Vector Space Model
LSI	Latent Semantic Indexing
XML	eXtensible Markup Language
TTL	Time-To-Live
CAN	Context Addressable Network
SDI	Selective Dissemination of Information
LS	Load-Shedding
DL	Digital Library

Chapter 1

Introduction

This thesis addresses the problem of offering scalable, adaptive, efficient, full-fledged information retrieval and filtering functionality in a peer-to-peer environment. In this introductory chapter, we define the problem, highlight our approach and present our contributions.

1.1 Problem Statement

Much information of interest to humans is today available on the Web. People can easily gain access to information but at the same time, they have to cope with the problem of information overload. Consequently, they have to rely on specialised tools and systems designed for searching, querying and retrieving information from the Web. Currently, Web search is controlled by a few search engines that are assigned the burden to follow this information explosion[1] by utilising centralised search infrastructures. Additionally, users are striving to stay informed by sifting through enormous amounts of new information, and by residing on tools and techniques that are not able to capture the dynamic nature of the Web. In this setting, peer-to-peer (P2P) Web search seems an ideal candidate that can offer adaptivity to high dynamics, scalability, resilience to failures and leverage the functionality of the traditional search engine with new features and services.

[1]Google alone indexed 11.2 billion web pages in November 2005, and 25.2 billion in May 2006. Google no longer publishes the number of indexed pages; these numbers were the search results for the query "$*$ $*$", which retrieves web pages containing any two words.

In P2P systems a very large number of autonomous computing nodes (the *peers*) pool together their resources and rely on each other for *data* and *services*. P2P networks have emerged as a natural way to share data, and have been popularised by applications such as file and cycle sharing, IP telephony etc. At the same time, P2P systems pose new challenges in designing distributed systems that go beyond the file sharing paradigm to full-fledged content search and Web search functionality. In this thesis, we focus on the problem of *P2P resource sharing in wide-area networks such as the Internet and the Web*. In the architecture that we envision, each peer owns resources which it is willing to share: documents, web pages or files which are appropriately annotated and queried using constructs from Information Retrieval models. There are two kinds of basic functionality that we expect this architecture to offer: information retrieval (IR) and information filtering (IF) (also known as publish/subscribe or information dissemination). In an IR scenario a user poses a query (e.g., "I am interested in papers on bio-informatics") and the system returns a list of pointers to matching resources. In an IF scenario, a user posts a subscription (or profile or continuous query) to the system to receive notifications whenever certain events of interest take place (e.g., when a paper on bio-informatics becomes available).

When trying to design and implement such a system there are several technical challenges that have to be faced. In this thesis, we look into the following four fundamental questions in the design of a distributed resource sharing system and provide state of the art solutions to each one of them:

- What are appropriate architectures to support IR and IF functionality in a P2P environment?
- What is an appropriate data model and a respective query language to annotate and query resources offered by the peers? Can it be designed in a principled and formal way?
- What are the protocols that regulate peer interactions and allow for the aforementioned functionality? Can we provide scalability and efficiency without sacrificing retrieval effectiveness?

- What are the local indexing algorithms that will allow each peer to manipulate large numbers of retrieval and filtering requests efficiently?

The rest of this thesis provides answers to the above questions and presents original results that advance the state of the art in each of the problems posed. Put together, our results form a framework for designing and implementing a distributed resource sharing system that supports information retrieval and filtering functionality using the P2P paradigm, and provide machinery that outperforms the best solutions found in the relevant literature.

1.2 Solution Outline

In this section, we provide an outline of our answers to the technical questions posed previously and argue that our approach provides interesting, state of the art solutions to each one of these fundamental challenges.

In the architecture we propose, nodes can implement any of the following types of services: super-peer service, provider service and client service. Nodes implementing the super-peer service (super-peers) form the message routing layer of the network. Each super-peer is responsible for serving a fraction of the clients by storing documents, indexing continuous queries, matching them against incoming (published) documents and creating notifications. The super-peers run a Distributed Hash Table (DHT) protocol which is an extension of Chord. A node implementing the client service (client) connects to the network through a single super-peer node, which is its access point. Clients can connect, disconnect or even leave the system silently at any time. Clients are information consumers: they can pose one-time queries to receive relevant resources, subscribe to resource publications with continuous queries and receive notifications about published resources (e.g., documents) that match their interests. Finally, the provider service (provider) is implemented by information sources that want to expose their contents to the clients of the system. A node implementing this service connects to the network through a super-peer which is its access point. To be able to implement this service, an information source creates meta-data for the documents it stores using an appropriate data model and pub-

lishes it to the rest of the network using its access point. Although our protocols are tailored for a two-tier architecture, our ideas can be easily applied in a pure P2P environment.

In the context of the proposed architecture, our main focus is on providing models and languages for expressing publications and subscriptions, protocols that regulate super-peer interactions and query indexing mechanisms that are utilized by each one of the super-peers.

The chosen data model and query language will have a serious effect on the DHT protocols, as the DHT is the layer in which publications, queries and subscriptions are indexed. We use a well-understood attribute-value model, called $\mathcal{AWPS}$, that is based on named attributes with free text as value interpreted under the Boolean and VSM (or LSI) models. The query language of $\mathcal{AWPS}$ allows Boolean combinations of comparisons $A\ op\ v$, where A is an attribute, v is a text value and op is one of the operators "equals", "contains" or "similar" ("equals" and "contains" are Boolean operators and "similar" is interpreted using the VSM or LSI model). We present a formal account of $\mathcal{AWPS}$ and its subsets $\mathcal{WP}$ and $\mathcal{AWP}$, and answer some standard questions from a logic and complexity perspective.

In the context of these models, we show how to provide IR and IF functionality by using an extension of the Chord DHT. To achieve this, we have designed and implemented a set of protocols, collectively called DHTrie, that extend the Chord protocols assuming that publications and subscriptions are expressed in the model $\mathcal{AWPS}$. Additionally, we showed experimentally that local data structures and simple routing optimisations can make a big difference in a DHT environment. The experiments showed that our protocols are scalable: the number of messages needed to publish a document and notify interested subscribers remains almost constant as the network grows. Moreover, the increase in message traffic shows little sensitivity to the increase in document size. Since probability distributions associated with publication and query elements are expected to be skewed in such a scenario, achieving a balanced load among the nodes becomes an important problem. Thus, we studied important cases of load balancing for DHTrie and presented a new algorithm, based on the idea of load-shedding, which is also applicable to the standard

DHT lookup problem.

In the architecture described above, clients subscribe to their access points with continuous queries that express their information needs, and providers expose their content using an appropriate meta-data model. Each super-peer is responsible for storing the queries so that whenever a resource is published, the continuous queries satisfying it are found and notifications are sent to the appropriate clients. A facet of this work deals with the filtering problem that needs to be solved efficiently by each super-peer: Given a database of continuous queries db and a document d, find all queries $q \in db$ that match d. We have proposed data structures and indexing algorithms that enable us to solve the filtering problem efficiently for large databases of queries expressed in the model $\mathcal{AWP}$, which is the Boolean subset of $\mathcal{AWPS}$ and is based on named attributes with values of type text, and word proximity operators. The algorithms presented here are used in combination with the algorithms of SIFT [210] that handle VSM queries to index the $\mathcal{AWPS}$ queries each peer is responsible for.

The main idea behind the local indexing algorithms is to store sets of words compactly by exploiting their common elements. In algorithm PrefixTrie, which is a modification of the most efficient indexing algorithm proposed in the literature, a query is considered as a sequence of words sorted in lexicographic order, and a trie is used to store queries compactly by exploiting common prefixes. Our proposed algorithm, coined BestFitTrie, constitutes an improvement over PrefixTrie. BestFitTrie keeps the main idea behind PrefixTrie but (a) handles the words contained in a query as a set rather than as a sorted sequence and (b) searches exhaustively the forest of tries to discover the best place to store a new query. This allows BestFitTrie to achieve better clustering of the queries and thus smaller filtering times. Finally we propose algorithm ReTrie, which improves over BestFitTrie by considering the periodic re-organisation of the query database.

1.3 Contributions

Early work in information retrieval and filtering over P2P networks focused on unstructured protocols (such as Gnutella and FastTrack). With the advent of dis-

tributed hash tables, a new wave of systems, that supported either kind of functionality by using the DHT as the routing substrate, appeared [10, 11, 35, 100, 171, 184, 187, 189]. Our work is the first approach in this area that makes the following important contributions:

- It studies the theory of models $\mathcal{WP}$ and $\mathcal{AWP}$ and focuses on questions related to satisfiability and entailment. These results have been published in [126].
- It shows how to support information retrieval and filtering functionality in a single unifying framework, using a DHT as the routing infrastructure. The results of this work are presented in [202].
- Contrary to other approaches in the area, this is the first work that aims for exact query answering (precision and recall are the same as those of a centralised system), while adopting a distributed architecture that provides scalability and efficiency benefits. These results have been published in [197, 199, 202]
- It extends the Chord protocols with pub/sub functionality, providing a tunable approach that targets low latency and low network load at the same time. In the same context it demonstrates the effectiveness of additional routing information, based on local interactions, in lowering message traffic and latency. These results have been published in [196, 201, 203].
- It proposes a solution of the local filtering problem that outperforms solutions in the current literature. Our trie-based query indexing algorithms are 20% faster than their counterparts, offering sophisticated clustering of user queries and mechanisms for the adaptive reorganisation of the query database when filtering performance drops [106, 125, 200].
- It applies these ideas in a digital library (DL) scenario and shows how to use the P2P paradigm to design future DLs. The results of this work are presented in [202].

To summarise the above, our approach offers a complete, state of the art suite of concepts, algorithms and tools that advance the state of the art and allow building a

distributed resource sharing system. This suite provides (i) data models and query languages for resource annotation and querying, (ii) scalable distributed protocols to support full-fledged retrieval and filtering functionality and (iii) efficient local indexing mechanisms. All these techniques have been applied to the digital library domain, in an effort to design and build a scalable, self-organising, low-maintenance architecture for DLs based on the P2P paradigm.

Distributed IR as studied here can benefit from techniques of traditional IR, distributed systems and networking (especially P2P networks), databases and distributed AI. Architectures and distributed protocols discussed in this work, advance the state of the art in the field towards self-organising approaches that are suitable for dynamic settings, and offer benefits such as adaptivity and failure resilience. These early efforts for full-fledged P2P information retrieval try to harness the properties of loosely coupled components to leverage the applicability of traditional IR.

Although many of the issues in this thesis were tackled with an IR perspective in mind, their application is not restricted to this domain. Content-based multicasting over structured overlays and load balancing in the presence of skewed item distributions are two examples of problems that are also interesting from a P2P perspective and this thesis makes a contribution to them. P2P research is lately dominated by endeavours to build new types of applications that will use overlay networks as their architectural paradigm. P2P Web search and P2P digital libraries are certainly two of the dominant and most interesting applications in this strand of research and this work lies within these efforts.

1.4 Thesis Structure

This thesis is comprised of seven chapters, the first being the current introductory chapter. The rest of the thesis is organised as follows. Chapter 2 positions the thesis with respect to related work by reviewing literature in the fields of P2P systems, information retrieval and information filtering. Chapter 3 presents the IR-based data models and the associated query languages that were used in this thesis and studies questions related to satisfiability and entailment in these models. Chapter 4 presents our P2P architecture, while Chapter 5 focuses on the case of information filtering

and presents an in-depth investigation of the pub/sub protocols that regulate peer interactions. Chapter 6 studies the filtering problem at each one of the peers and proposes data structures and algorithms that solve it efficiently. Finally, Chapter 7 summarises the achievements of this thesis and indicates directions for future research.

Chapter 2

Related Research

In this chapter we provide an overview of previous research which is relevant to the topics of this thesis. Initially, we focus on architectural aspects of P2P networks, and present a comprehensive survey of P2P networks, ranging from unstructured approaches to super-peers and structured overlays. Then, we present related work in the areas of IR and IF using P2P networks.

2.1 Peer-to-Peer Networks

In P2P systems a very large number of autonomous computing nodes (the peers) pool together their resources and rely on each other for data and services. P2P networks have emerged as a natural way to share data. Popular systems such as Napster[1] (now in a commercial service), Gnutella[2], Freenet[3], Kazaa[4], Morpheus[5] and others have made this model of interaction popular. Ideas from P2P computing can also be applied to other distributed applications beyond data sharing such as Grid computation (e.g., SETI@Home[6] or DataSynapse[7]), collaboration networks

[1] `http://www.napster.com`

[2] See `http://www.limewire.com` for one of the various clients implementing the Gnutella protocol or its variations.

[3] `http://freenet.sourceforge.net`

[4] `http://www.kazaa.com`

[5] `http://www.musiccity.com`

[6] `http://www.setiathome.ssl.berkeley.edu`

[7] `http://www.datasynapse.com`

(e.g., Groove[8]), IP telephony (e.g., Skype[9]) and even new ways to design Internet infrastructure that supports sophisticated patterns of communication and mobility [181].

P2P networks are typically distinguished into three different classes according to their topology: unstructured networks, structured networks and hierarchical networks. In unstructured networks all peers are equal and form an overlay network with no restrictions on topology and no centralised source of information. Gnutella is considered the prototypical *symmetric* or *unstructured* P2P network. Since its original proposal, the inefficiencies of this basic Gnutella protocol have carefully being studied and various proposals for more efficient search in unstructured P2P networks are now in the literature (see [204] for a recent comparison). Structured networks, on the other hand, have a regular topology, e.g. rings or hypercubes, and were devised as a remedy for the routing and object location inefficiencies of unstructured networks. DHTs are a prominent class of structured overlays that attempt to solve the object lookup problem by offering some form of distributed hash table functionality: assuming that data items can be identified using *unique numeric keys*, DHT nodes cooperate to store keys for each other. Finally, hierarchical networks partition the nodes into two sets: super-peers and clients. In a super-peer system, all super-peers are equal and have the same responsibilities. Each super-peer serves a fraction of the clients and keeps indices on the resources of those clients. Super-peers interact by following a protocol of their choice (e.g., a symmetric one like Gnutella, a structured one like Napster[10] or a DHT protocol). Clients can run on user computers and resources (e.g., files in a file-sharing application) are kept at client nodes. Clients are equal to each other since the software running at each client node is equivalent in functionality. Clients learn about resources by querying super-peers and download resources directly from other clients.

In the rest of this section we discuss the first three systems that popularized the P2P paradigm: Napster, Gnutella and Freenet and introduce super-peer networks.

[8] `http://www.groove.net`

[9] `http://www.skype.com`

[10] Napster can also be seen as a *structured* P2P system: each node has well-defined information about other nodes in the system [25]. However, centralized systems such as Napster have well-understood problems of scalability and resilience.

Finally, we briefly survey some known DHTs and focus on Chord, an early influential DHT that forms the basis of our work.

2.1.1 Three Influential Peer-to-Peer Systems

The common goal of the three pioneer systems, Napster, Gnutella and Freenet, was to facilitate the discovery and sharing of files (e.g., images, audio and video) among a large set of *peers* (user computers) located at the "edge of the Internet". The files to be shared are stored at the peers, and after being discovered by an interested party, they are downloaded using a protocol similar to HTTP. But beyond this basic goal, there are important differences among the three systems regarding the *metadata* kept at each network node, the *topology* of the P2P network, the *placement* of the shared files, the *routing algorithms* for queries and replies, the degree of privacy offered to its users, etc.

Napster

In Napster (shown in Figure 2.1), a large cluster of dedicated servers owned by the respective company maintains a *metadata index* that keeps track of active peers, descriptions of the files they are willing to share, and certain quality of service parameters such as bandwidth and duration of the connection. Peers (clients!) connect to the system by connecting to *one* of these dedicated servers and publish a description of the files they want to share with other peers. Peer queries are sent to their selected server which returns a list of matching files, the address of the peer which has each file, and other important information known about the peer (e.g., bandwidth as reported by the peer). Peers can then *directly* download selected files from the peer of their choice. Thus, Napster is a *hybrid* P2P system [214] with some client-server features (register, publish, query) and a single P2P feature (download). Napster is also a *structured* P2P system: each node has well-defined information about other nodes in the system [25]. However, centralised systems such as Napster have well-understood problems of scalability and resilience (so one has to use techniques such as indexing for scalability and replication for resilience [89]).

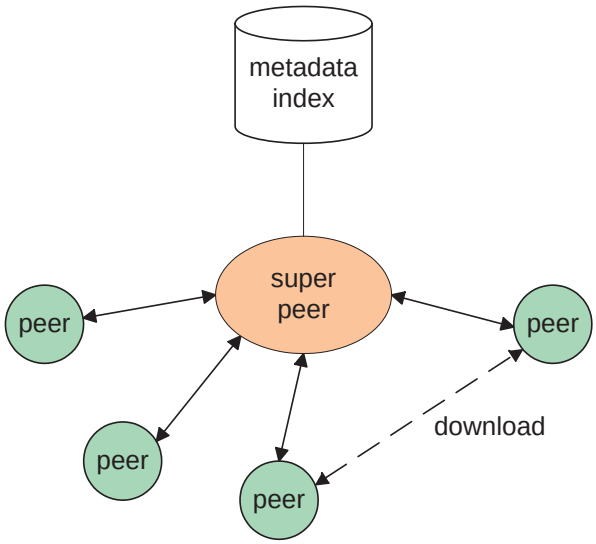


Figure 2.1: An example of the Napster P2P system

Gnutella

On the opposite end of the spectrum of decentralization, Gnutella (shown in Figure 2.2) has a *symmetric protocol* and *no centralized servers*. Each node in a Gnutella network is a peer that can act as a client and as a server at the same time. Gnutella peers form an *overlay network* by setting up connections to peers of their choice. Addresses for connecting to the Gnutella network initially can be found by the interested user (e.g., by consulting web pages such as `gnutellahosts.com` or `router.limewire.com`). Gnutella offers primitives *ping* and *pong* for discovering parts of the network and facilitate its maintenance while peers enter and leave the system.

To discover a file, a Gnutella source node issues a query such as "I am interested in MPEGs with video-clips of Jennifer Lopez" to its neighbors with whom it has open connections. The query is accompanied by a time-to-live (TTL) counter that specifies how many hops this query is allowed to travel in the Gnutella network. Each node that receives this request processes it using its local file collection and returns URLs pointing to matching files to the requesting node. Then, this node decrements the TTL counter of the request by one. If the value of the TTL counter is greater than 0, then this node forwards the query to its neighbours. This process is repeated and eventually more pointers to matching files are returned to the source of the

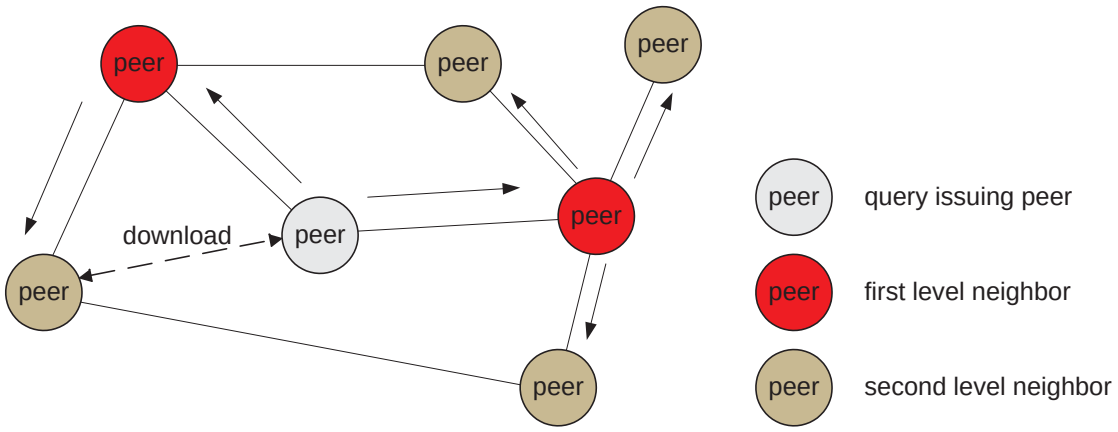


Figure 2.2: An example of the Gnutella P2P system

request. Thus, Gnutella uses the classical *flooding* technique known from Computer Networks [36] for routing queries. Replies reach the source node by travelling along the reverse path followed by the query, as a node precessing a request in a Gnutella network does not know the identity of the source node issuing this request. However, the privacy of information requesters and providers is not really protected in any serious way (e.g., Gnutella messages contain IP addresses, and URLs are returned to information requesters so that they can retrieve the files they desire).

Gnutella is considered the prototypical *symmetric* or *unstructured* P2P network. Since its original proposal, the inefficiencies of this basic Gnutella protocol have carefully being studied and various proposals for more efficient search in unstructured P2P networks are now in the literature (see [204] for a recent comparison).

Freenet

Freenet is a P2P network of nodes connected to each other for the purpose of sharing information in the form of data files [56]. Like Gnutella, Freenet keeps a *completely decentralised architecture* which ensures scalability, robustness and fault-tolerance. At the same time, Freenet invests effort in ensuring the survivability of published information, the adaptability to usage patterns and the protection of the anonymity of information providers, consumers and holders (these features are what distinguish Freenet from Napster and Gnutella).

Every Freenet user runs a node that provides the network with some storage

space. To add a file, a user sends the network a message containing the file and an assigned location-independent *globally unique identifier (GUID)* which is computed using a SHA-1 [153] secure hashing function. Each GUID consists of two parts: a *content-hash key* which is obtained by hashing the contents of the file and it is used for low-level data storage, and a *signed-subspace key* intended for higher-level human use like traditional filenames.

To retrieve data, a user of Freenet sends to the network a request message and the GUID of the file. Whenever a node receives a request, checks its local data-store first. If the file is found, the node returns it to the requester together with a tag identifying itself as the holder. If the file is not found, the routing table is consulted, *one* of the neighbour nodes with the closest matching key is chosen, and the request is forwarded to it. This is a basic difference with the Gnutella algorithm: Gnutella does not perform *heuristic* search and would send the query to *all* neighbour nodes. When the data file is finally found, it is returned to the requester via the same path. Additionally, intermediate nodes save an entry in their routing table associating the requested key with the data source. Depending on their distance from the holder, each node might also cache a local copy of the data file.

The anonymity of a file producer is ensured by having intermediate nodes occasionally altering the holder tags to point to themselves as data holders. This does not compromise discovery of the file later because the identity of the true data holder is kept at the node's routing table and routing tables are never revealed. The anonymity of an initiator of a query is also ensured since a node cannot know whether its neighbour node is the one interested in the results of the query or is simply forwarding a message.

Each data request in Freenet is given a TTL count (like in Gnutella), which is decremented at each node the request goes through successfully in order to reduce message traffic. To prevent requests from going into an infinite loop, Freenet assigns a unique identifier to each request so that a node will never forward a request that goes through it for a second time.

Insert messages follow the same procedure that a request message for that file would follow, thus routing tables are updated in the same way and files are stored

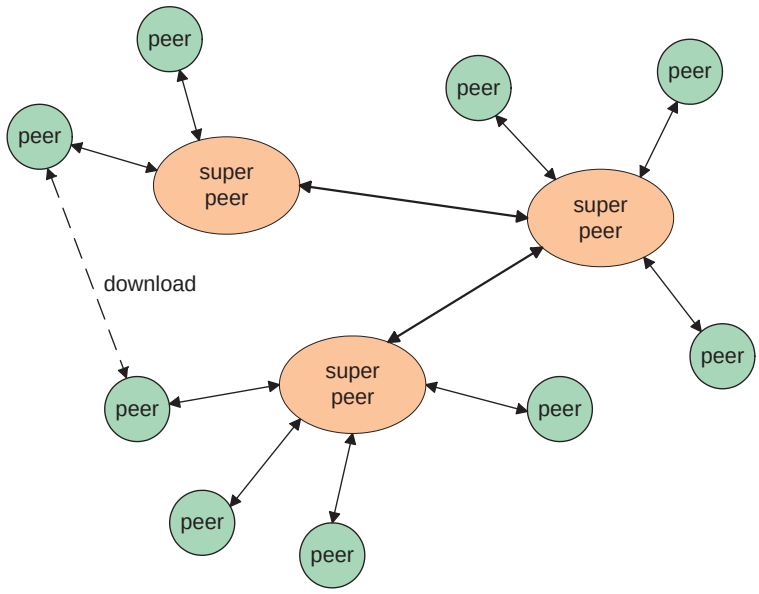


Figure 2.3: An example of a super-peer network

in exactly the nodes where queries will go looking for them [56].

2.1.2 Super-Peer Networks

One approach to deal with the scalability of Gnutella-like systems is to introduce a hierarchy of peers (as shown in Figure 2.3): *super-peers* (or *ultra-peers*, or *hubs*) and *clients* (or simply peers). This was the approach pioneered by the P2P platform Fast-Track and is being used in deployed systems such as Kazaa, Morpheus and new versions of Gnutella. In a super-peer system, all super-peers are equal and have the same responsibilities. Each super-peer serves a fraction of the clients and keeps indices on the resources of those clients. Super-peers interact by following a protocol of their choice (e.g., a symmetric one like Gnutella, a structured one like Napster or a DHT protocol). Clients can run on user computers, and resources (e.g., files in a file-sharing application) are kept at client nodes. Clients are equal to each other since the software running at each client node is equivalent in functionality. Clients learn about resources by querying super-peers and download resources directly from other clients.

The issues involved in the design and implementation of super-peer systems have recently been studied in academia as well. One of the first works to study

the characteristics of super-peer networks and present the performance tradeoffs associated with them was [212]. Super-peers are generally either volunteers that want to offer their extra resources to the community or are chosen and "promoted" by some mechanism (e.g., a gossiping mechanism as in [110]). The nice properties of super-peer networks made then an attractive architectural solution adopted by a number of systems and research proposals such as Edutella [149], P2P-DIET [54, 106] and also [130, 132, 168].

2.1.3 Distributed Hash Tables

The success of P2P protocols and applications, such as Napster and Gnutella, motivated researchers from the distributed systems, networking and database communities to look more closely into the core mechanisms of these systems and investigate how these could be supported in a principled way. This quest gave rise to a new wave of distributed protocols, collectively called distributed hash tables, that were aimed primarily at the development of P2P applications [1, 15, 97, 135, 165, 173, 180]. DHTs are *structured* P2P systems, which attempt to solve the following *look-up problem*:

> Let X be some data item stored at some distributed dynamic network of nodes. Find data item X.

The core idea in all DHTs is to solve this look-up problem by offering some form of distributed hash table functionality: assuming that data items can be identified using *unique numeric keys*, DHT nodes cooperate to store keys for each other (data items can be actual data or pointers). Implementations of DHTs offer a very simple interface consisting of two operations:

- `put(ID, item)`. This operation inserts item with key `ID` and value `item` in the DHT.
- `get(ID)`. This operation returns a pointer to the DHT node responsible for key `ID`.

Although the DHTs available in the literature differ in their technical details, all of them address the following central questions:

- *How do we map keys to nodes?* Keys and nodes are identified by a binary number. Keys are stored at one or more nodes with identifiers "close" to the key identifier in the identifier space.
- *How do we route queries for keys?* Any node that receives a query for key k, returns the data item X associated with k if it owns k, otherwise it forwards k to a node with identifier "closer" to k using only local information.
- *How do we deal with dynamicity?* DHTs are able to adapt to node joins, leaves and failures and update routing tables with little effort.

The answers to the above questions can give us a good high-level categorization of existing DHTs [17, 25]. Although the effort of [6] is on providing a reference model to unify different P2P approaches, it makes a good job on distinguishing six basic concepts common on all structured overlays that can be used to categorise them. Among these concepts routing strategy, maintenance, identifier space management verify our categorisation schema based on the previously asked questions. Here we discuss the most popular DHTs and present variants and other hybrid approaches that try to harness the benefits of both unstructured and structured networks.

Scalable Distributed Data Structures

The roots of DHTs are traced back to hashing algorithms and protocols over clusters, in work conducted some years ago in the area of distributed data structures and databases. That time the term Scalable Distributed Data Structures (SDDS) was coined by the Litwin and colleagues in [129] to introduce a class of data structures used in a distributed environment. Although the principles of SDDS and DHTs are the same, the application area of the two approaches is very different, since DHTs were designed and deployed in highly dynamic environments with loosely coupled components. On the other hand, Linear Hashing (LH) [128] was initially designed for a single site of a multiprocessor environment with shared memory. In this setting,

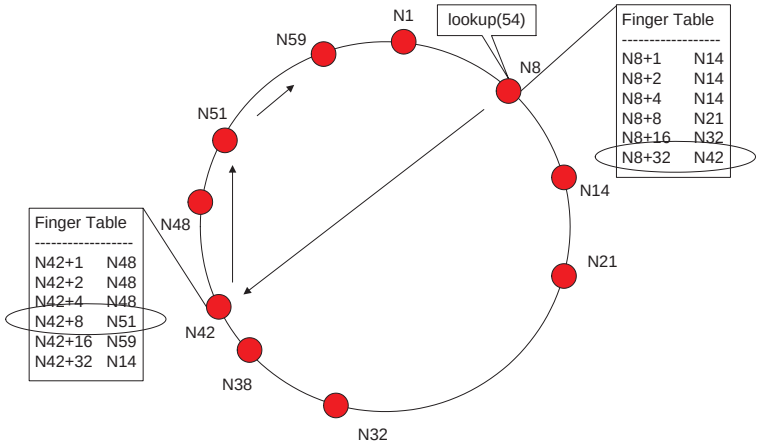


Figure 2.4: An example of a lookup operation over a Chord ring with m=6

LH provides a hashing method that allows the address space to grow or shrink dynamically and allows disk files to change size dynamically without deterioration in space utilisation or access time. Its extension, LH* [129] can accommodate any number of clients and servers and allows a file to extend to any number of sites. It does not require a central coordinator and records can be located at a constant number of hops independently of the number on sites.

Chord

Chord [180, 182] uses a variation of *consistent hashing* [112] to map keys to nodes. In the consistent hashing scheme each node and data item is assigned an m-bit identifier, where m should be large enough to diminish the possibility of different items hashing to the same identifier (a cryptographic hash function such as SHA-1 is used). The identifier of a node can be computed by hashing its IP address. For data items, we first decide a key and then hash it to obtain an identifier. For example, in a file-sharing application the name of the file can be the key (this is an application-specific decision). Identifiers are ordered in an *identifier circle (ring)* modulo 2^m i.e., from 0 to $2^m - 1$. Figure 2.4 shows an example of an identifier circle with 64 identifiers ($m = 6$) and 10 nodes.

Keys are mapped to nodes in the identifier circle as follows. Let H be the consistent hash function used. Key k is assigned to the first node which is equal or follows $H(k)$ clockwise in the identifier space. This node is called the *successor*

node of identifier $H(k)$ and is denoted by $successor(H(k))$. We will often say that this node is *responsible* for key k. For example in the network shown in Figure 2.4, a key with identifier 30 would be stored at node $N32$. In fact node $N32$ would be responsible for all keys with identifiers in the interval $(21, 32]$.

If each node knows its successor, a query for locating the node responsible for a key k can always be answered in $O(N)$ steps where N is the number of nodes in the network. To improve this bound, Chord maintains at each node a routing table, called the *finger table*, with at most m entries. Each entry i in the finger table of node n, points to the first node s on the identifier circle that succeeds identifier $H(n)+2^{i-1}$. These nodes (i.e., $successor(H(n)+2^{i-1})$ for $1 \leq i \leq m$) are called the *fingers* of node n. Since fingers point at repeatedly doubling distances away from n, they can speed-up search for locating the node responsible for a key k. If the finger tables have size $O(\log N)$, then finding a successor of a node n can be done in $O(\log N)$ steps with high probability [180].

To simplify joins and leaves, each node n maintains a pointer to its *predecessor* node i.e., the first node *counter-clockwise* in the identifier circle starting from n. When a node n wants to join a Chord network, it finds a node n' that is already in the network using some out-of-band means, and then asks n' to help n find its position in the network by discovering n's successor [182]. Every node runs a *stabilization* algorithm periodically to learn about nodes that have recently joined the network. When n runs the stabilization algorithm, it asks its successor for the successor's predecessor p. If p has recently joined the network then it might end-up becoming n's successor. Each node n periodically runs two additional algorithms to check that its finger table and predecessor pointer is correct [182]. Stabilization operations may affect queries by rendering them slower (because successor pointers are correct but finger table entries are inaccurate) or even incorrect (when successor pointers are inaccurate). However, assuming that successor pointers are correct and the time it takes to correct finger tables is less than the time it takes for the network to double in size, one can prove that queries can still be answered correctly in $O(\log N)$ steps with high probability [182]. In [180] the details of the Chord protocols for node joins and leaves, stabilisation and fault-tolerance are provided.

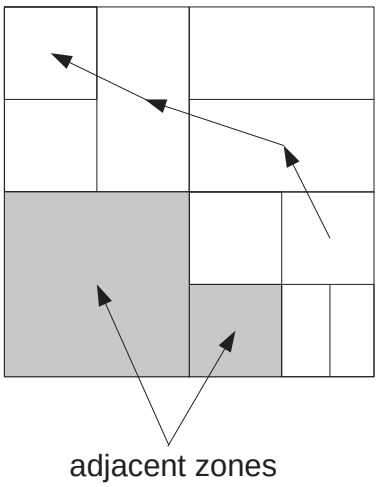


Figure 2.5: An example of query routing in a CAN DHT with d=2

To deal with node failures and increase robustness, each Chord node n maintains a *successor list* of size r which contains n's first r successors. This list is used when the successor of n has failed. In practice even small values of r are enough to achieve robustness [182]. If a node chooses to leave a Chord network voluntarily then it can inform its successor and predecessor so they can modify their pointers and, additionally, it can transfer its keys to its successor. It can be shown that with high probability, any node joining or leaving a Chord network can use $O(\log^2 N)$ messages to make all successor pointers, predecessor pointers and finger tables correct [180].

CAN

The concept of a Content-Addressable Network (CAN) is introduced in [165], where a distributed infrastructure that provides hash table functionality is introduced. In CAN, a Cartesian coordinate space consisting of d dimensions is partitioned into *zones*. Each node n is responsible for one zone and maintains information about zones that are *adjacent* to his. As in any DHT, the coordinate space is used to store key-value pairs. To store a pair (k, v), the key k is hashed using a uniform hash function $H()$ to produce a point p in the coordinate space such that $p = H(k)$. The corresponding pair is then stored in the node that is responsible for the zone within which point p lies.

If a node n wants to retrieve a key-value pair (k, v), the same deterministic

hash function $H()$ is used to produce a point p in the coordinate space such that $p = H(k)$. If p does not lie in n's zone or in any of the zones of its immediate neighbours, the request must be routed through the overlay network until it reaches node n' that owns the zone within p lies. Message forwarding takes $O(d\sqrt[d]{N})$, where N is the number of nodes in the network, if a greedy routing approach that sends the message to the neighbour closest to the target zone is utilised. With $d = \log N$, each node can be reached in $O(\log N)$ hops and the routing table in each node takes $O(\log N)$ space.

To join CAN, a node n picks a random point p in the coordinate space (e.g., produced by hashing its IP address and port number), and contacts node n' that is responsible for the zone within p lies. Upon contact, node n' splits its zone in half and assigns one half to n. The key-value pairs lying in the zone to be handed over are also transferred to n. Splitting of zones is done by assuming a zone ordering and applying a round robin algorithm so that zones can be re-merged when nodes leave. Departures or failures of nodes are handled in a similar way by merging zones of the failed neighbours. Additionally node joins and leaves are cheaper compared to other DHT approaches (e.g., Chord) since the number of neighbours a node maintains depends only on the dimensionality of the coordinate space and not on the number of nodes in the system.

[165] mentions various optimisation to tackle important problems appearing in CAN. These optimisations include the incorporation of multiple coordinate spaces to deal with data availability, the use of multiple hash functions and peers per zone to deal with fault-tolerance, and the use of improved routing metrics to deal with routing efficiency. An extension of CAN that tackles the problem of application-level multicast is presented in [166], where multicast functionality can scale to large groups of nodes at trivial extra cost due to the CAN infrastructure.

Pastry

In [162] a distributed algorithm for accessing resources in an overlay network is presented by Plaxton and colleagues. This algorithm uses *suffix-based hypercube routing* to efficiently locate a resource by utilising routing tables of small size stored

in each node in the network.

Pastry [173] uses a version of Plaxton's algorithm as the core of its routing mechanism, modified appropriately to be used in a dynamic environment. Assuming a network of N nodes, a routing table stored at node n contains $\log N$ rows of entries. Each entry of a row k stores the IPs of those nodes, the identifier of which shares the same k bits with the identifier of n, but is different in bit $k+1$. Should there be more nodes that satisfy this criterion that a row is intended to store, a proximity metric is used to select the nodes closer to n. Apart from the routing table, a Pastry node also contains a neighborhood set and a leaf set. The neighborhood set contains contact information about the nodes closest to n, whereas the leaf set contains the numerically closest to n larger and smaller identifiers in equal percentage.

A message in Pastry is routed using prefix-based routing contrary to Plaxton's algorithm and Tapestry described shortly in the next section. Message forwarding in Pastry is done as follows. The node checks its leaf set to see if the key falls in this range. If so, the message is forwarded to the node with an identifier that is closest to the key. If not, the routing table is used and the receiver is the node that shares a common prefix with the key by at least one more digit. In special cases where no such entry is available in the routing table, the message is forwarded to another node only if it is numerically closer to the key than the current node and shares a prefix with the key at least as long as the current node. If the routing tables and leaf sets are correct, the expected number of hops to route a request to the responsible node is at most $\lceil \log_{2^b} N \rceil$, where b is an algorithm parameter typically set to 4.

Tapestry

Tapestry [98] bears many resemblances to Pastry and its routing mechanism is also modelled after Plaxton's algorithm, with the main differences focusing on adaptability, fault-tolerance and other optimisations. Routing is suffix-based and failures are detected using timeouts and a heartbeat mechanism. Contrary to Plaxton's work, Tapestry assigns multiple roots (responsible nodes) for each resource by concatenating seed values (e.g., 1, 2, 3) with resource identifiers. Also, to incorporate fault tolerance, Tapestry maintains multiple neighbors per routing table entry. Finally, a

distinguishing feature of Tapestry is its ability to take node proximity into account. As a message travels around the network, it is preferable to be routed through nodes that are close (from the physical network point of view) to each other.

Support for dynamic insertion/deletion of resources and protocols for join/departure of nodes and support for mobile resources is also provided by Tapestry. A dynamic environment makes the join/leave operations more complex, since proximity has to be taken into account. A detailed description of these features is out of the scope of this survey and the interested reader is referred to [98].

P-Grid

P-Grid [1, 3, 7] is a routing infrastructure based on a distributed search trie. Nodes in P-Grid are self-organised into a distributed trie by pairwise interactions, allowing the data structure to adapt to different key distributions. Each node holds a part of the distributed trie, and its position is determined by a bit string that represents the information the node is responsible for. Prefix-based routing is utilised to locate a data item stored at a node. To facilitate routing, each node maintains a reference to at least one other node that is responsible for the other part of the trie at the same level. When a search request is issued there are two possibilities for each node; either the node is responsible itself for forwarding the request to the next level, or the request has to be forwarded to another peer responsible for the other part of the trie at the same level.

One of the key characteristics of P-Grid is its ability to modify the routing infrastructure depending on the key distribution, using path extension or retraction. This results to the modification of the shape of the underlying (virtual) trie, achieving uniform load distribution for nodes. Prefix and range search in this DHT approach is easy to do due to the trie organisation of the underlying network [65].

Other DHTs

In this section, a number of Chord variants and other DHT approaches are presented. Kademlia [139] uses consistent hashing of keys and nodes to place them in a metric

space. Key-value pairs are stored in nodes with identifiers that are close to the identifier of the key, using the XOR value of the identifiers as the distance metric. This offers a different notion of closeness, which makes the distance between any two nodes symmetric. This results in allowing the node to chose its neighbours from a set of available choices, which in turn benefits routing (by reducing e.g., latency) and fault tolerance. Fault tolerance is also supported by avoiding timeouts to detect failed nodes.

Viceroy [135] maintains a constant-degree, logarithmic approximation of a butterfly network [99] for routing. A Viceroy network contains three sets of links: (i) links for a Chord-like ring, coined *general ring*, where a node is connected to its ancestor and predecessor, (ii) links for a *level ring*, where nodes at the same level are connected in a ring and (iii) butterfly links, where two "down" links and one "up" link that connect a node with lower and upper level nodes are utilised. This constant degree (7 links) routing table achieves routing in a logarithmic number of hops, which made Viceroy the first DHT to achieve this. For a node join, $O(1)$ nodes need to change their state and parallel joins and leaves can be handled effectively. However nodes failures and fault tolerance in general in a constant degree network are open issues.

Koorde [111] is a Chord variant that inherits the simplicity of Chord and uses de Bruijn graphs [66] to route messages in a logarithmic number of hops by maintaining constant-degree routing tables. Like Chord, Koorde uses consistent hashing to map keys to nodes. The Koorde approach embeds de Bruijn graphs on a sparsely populated identifier ring (since way less than 2^{128} nodes are online each moment) and uses only 2 links per node n; the successor of n and the first node in the de Bruijn graph that n belongs to. One of the major contributions of Koorde is the tunable performance of hop counts per query that depends of the out-degree for each node.

Symphony [136] uses a probabilistic approach to form a Chord-like distributed data structure. The probability of a link between two nodes in Symphony is inversely proportional to the distance between them. Simulation results show that the system scales well for large number of nodes and updates are relatively cheap, providing an interesting alternative to previously proposed routing schemes.

Systems that try to harness the benefits of both unstructured and structured networks have also been proposed. Kelips [94] is a super-peer network that uses a DHT approach to organise nodes into affinity groups. Each node maintains information about all other nodes in the group (a prohibitively high cost for large network) and one node from each one of the other groups. Gossiping algorithms (in the spirit of [8, 70, 108, 109, 144, 205]) are used for maintaining network information. Similarly Yappers [87] uses a Gnutella-like network and consistent hashing for key storage. Nearby nodes construct small DHTs and these DHTs form an unstructured network. In this way, complete reorganisation of the overlay is avoided and topology changes have a local effect, while the performance of the Gnutella protocol is improved. Finally SPROUT [138] proposes a DHT routing algorithm that is based on social links to leverage trust in an structured P2P network.

HyperCuP [177] offers a P2P topology that features a logarithmic network diameter without relying on uniform hash functions. It organises nodes in a hypercube graph with a tunable base b that is used to adjust the node's out-degree and network diameter. Nodes are able to join and leave at any time, and other nodes in the network take up their place to complete the hypercube graph. This design offers efficient broadcast and search functionality especially for multi-point search, contrary to other DHT approaches that need specialised algorithms to achieve multicasting [102, 166].

$DKS(N, k, f)$ [15] is a family of routing infrastructures using three parameters (N - the maximum number of nodes, k - the search arity and f - the degree of fault-tolerance) to create an instance of an overlay network. As the authors state, $DKS(N, k, f)$ can be seen as a generalisation of Chord that however does not use the active correction mechanism, but rather rely on correction-on-use. Since active correction mechanisms have been shown to be a significant percentage of the message load incurred by a DHT, considering out-of-date routing entries as the normal situation rather than an abnormality is a promising research direction. In $DKS(N, k, f)$ the lack of active correction of nodes' routing tables is based on the observation that the data insertion and lookup messages are sufficient in number so that they can be used to carry out the maintenance of the overlay as well.

Another feature that makes $DKS(N, k, f)$ a Chord generalisation is the arity used in the distributed search mechanism. The search arity k is used to divide the search space in k equal parts. The larger the search arity is, the larger are the routing table entries of the nodes and thus the smaller the lookup length. When $k = 2$, the data structures and the lookup performance are similar to those of Chord.

All DHT designs proposed over the last few years assumed a non-hierarchical structure of the overlay. Starting from this observation, [88] proposes Canon, a hierarchical DHT that tries to combine the benefits of both flat and hierarchical designs. The Canon paradigm offers the same routing facilities as a standard flat DHT, while at the same time provides efficient multicasting, fault isolation and better adaptation to the underlying network. To show the benefits of Canon, the authors apply their design to four popular DHTs and produce their hierarchical versions which they name after a grotesque paraphrasing of their original names.

Some research has recently been directed to DHT systems that are resilient to failures; either these failures are random or directed from an adversary that possesses some knowledge of the system state. A simple approach with logarithmic linkage, load and dilation, which can guarantee that almost all nodes can locate almost all items under hostile situations, is presented in [147]. In this work random failures and random spam generation (a node generating arbitrarily false versions of a requested data item) are addressed. The authors however do not address the problem of adversarial node deletion. In [77] this problem is addressed through the adoption of a butterfly network. In this work a network that is resistant to the deletion of a constant fraction of the nodes is described, giving guarantees that most of the nodes will be able to access a large percentage of the data. The authors report that such a network is able to operate even if half of the nodes go down.

Tornado [101] is a capability-aware DHT design that accommodates for node heterogeneity. Each node according to its computational capacity and willingness to contribute decides the number of data item sets (called virtual homes) to store. Based on a distributed k-ary search tree, Tornado offers $O(\log_k N)$ lookup time, with N being the number of nodes in the network. A notion of node quality is also introduced, by the distinction between “good” and “bad” peers which is mainly

made upon the resources they are willing to contribute.

Most of the approaches of DHTs discussed here use some form of hashing to give nodes and data a unique identity; in practise an overlay graph in the form of a metric space is built and both data and nodes are placed in this space. Based on this placement, a resource is assigned to a node using some mapping scheme. A cryptographic hash function offers a uniformly populated overlay space and a uniform assignment of resources to nodes. While this is a desirable property, it has the disadvantage of destroying locality, that is an important feature in many applications. Some DHT-based approaches have considered this problem [38, 161, 194] and resorted to solutions such as order-preserving hashing. Another strand of research was taken by two independent approaches; SkipNet [96] based on skip lists and [22] based on skip graphs. Skip lists are augmented ordered linked lists with forward links to allow for skipping large parts of the list while searching. They are a probabilistic data structure with efficiency comparable to that of a binary tree. SkipNet builds a trie of skip lists that are organised as a ring (the last item in the list points to the first). SkipNet favours path locality by ensuring that the traffic within a domain travels only in nodes that belong to this domain. Additionally each node stores its own data, which preserves locality. [22] proposes a similar distributed data structure where node additions and deletions can be done in time that is logarithmic to the size of the graph. The properties of skip lists and the lack of a hashing mechanism provides data locality and range query support is inherent in these approaches.

A distributed trie approach is presented in [84], where the keys are maintained in a trie that is distributed among the nodes. Stale resources are updated by piggybacking appropriate information on query massages that travel the network, thus coping only with resources that are requested often. Nodes learn system state by local interaction and system converges to an accurate network map. However stale views of peers may result in broadcasting of a lookup message to a large fraction of nodes in the network.

2.2 Applications of Peer-to-Peer Networks

The P2P computing paradigm has been proposed and used for a number of diverse applications in many scientific areas. File sharing and IP telephony are definitely two of the most prominent examples of such applications that made the P2P computing model popular to the crowds and also to the scientists. Grid computation was also a popular application of P2P computing advertised by projects such as SETI@Home and DataSynapse. Moving on a different direction, DHash [63] proposes the use of a structured P2P network to implement an alternative to the DNS service, and in the same spirit Overlook [191] and [26] use the P2P paradigm to provide distributed name services. Network performance measurements are another interesting application scenario built on top of P2P networks. In [179] a distributed system for network performance monitoring is proposed, and peers distributed all over the world are used to provide measurements increasing the network coverage of any manually selected set of measurement points.

The success of file sharing brought to light an interesting application that received attention from various research communities in the world. Distributed information (or resource) management emerged as an interesting research area that could benefit from the P2P paradigm. The idea behind this class of applications was simple yet fundamental; peers make their resources available to the network and the users query the network to retrieve resources that match their interests. Resources can be data that use a specific data model (relational, RDF, XML) or services (described using an appropriate service description language). In this work we put our focus in the problem of distributed resource sharing and in our case the data shared by the peers are documents and document metadata represented with some appropriate data model, and users query the network using a query language based on IR concepts.

Building such a system involves a number of important issues that need to be addressed, with system architecture and data model to be two of the most important design parameters. In the following sections we survey related research on systems that use an IR-based data model to represent their resources and categorise these approaches using their architectural choices.

2.3 Information Retrieval in Peer-to-Peer Networks

In this section, we survey related work in the context of information retrieval in P2P networks. Early works in the area involve IR on top of unstructured networks, but with the emergence of DHTs, solutions that exploit the routing efficiency of such networks have appeared.

P2P networks seem to be an interesting architectural solution for large scale information retrieval systems due to inherent features such as autonomy, scalability and fault-tolerance. Work in the area of P2P IR [27, 155, 156] focuses on building efficient mechanisms that can support advanced retrieval models and languages in the dynamic setting of a P2P system.

2.3.1 Information Retrieval in Unstructured Peer-to-Peer Networks

Early work in information retrieval over P2P networks focused on unstructured protocols (such as Gnutella and FastTrack). In this area of work, the query issued by the user floods the network with a time-to-live (TTL) restriction, and answers are collected and returned to the querying peer. This approach has the obvious inefficiency of probing a large set of peers unrelated to (or with a few information about) the query.

Keyword Search

Some of the early efforts to support IR functionality on top of unstructured P2P systems focused on simple keyword search. In [213], selective query forwarding based on aggregated statistics is examined, and a heuristic that takes into account the number of answers returned from each peer is reported to provide the most promising results. In this heuristic, the query is forwarded to the top k peers that returned the most results in the last ten queries. In the same philosophy, [219] proposes a similar technique, but with the usage of simpler peer statistics and with a focus on the quality of the returned results, unlike [213] where the focus is more

on the quantitative aspect.

To improve efficiency in locating rare items in unstructured networks, [57] uses associations inherent to everyday life (e.g., champagne and caviar are often bought together) to built associative overlays. In this architecture probing of peers irrelevant to a query is avoided by grouping similar peers into clusters and routing queries about rare items to the most relevant cluster. Although many works have considered improving existing distributed IR algorithms and adapting them to the P2P setting, the work of [218] showed that topology-aware techniques that take advantage of the network characteristics can improve the accuracy and performance of current IR techniques.

Vector Space Model

Advances in query routing in unstructured networks lead the research community to adopt more sophisticated data models and query languages. In PIRS [216] the authors use careful propagation of metadata information in a Gnutella-style network to be able to answer VSM queries in highly dynamic environments. Thus, PIRS tries to improve query result quality and scalability by treating metadata as an dynamic resource managed by all the peers in the system. This is achieved through careful metadata collection, heuristic distribution among the peers and IR-style ranking. In a more recent work [215], the authors try to reveal how individual characteristics (such as churn, metadata quality etc.) of current P2P systems affect the quality of the returned results. Thus, they consider different ranking functions and metadata description techniques and measure the effectiveness of each combination.

The PlanetP [64] system uses a variation of *tf/idf* to decide what nodes should be contacted to answer a query. To facilitate this, each peer actively disseminates its inverted index using compact summaries acquired through a Bloom filter and a gossiping algorithm to advertise its contents to the network. [146] reports that although PlanetP works well for moderately-sized P2P networks, it is not scalable to big network sizes with churn. This is because the number of summaries grows linearly to the number of peers in the network, and churn makes summary gossiping even more bandwidth-consuming. Thus, [146] proposes Rumorama, a framework

where a PlanetP network is divided to subnetworks, and a peer queries representatives of each subnetwork to get an answer set. A similar approach that tries to create clusters of nodes with similar interests is presented in [164], where a distributed peer clustering mechanism is proposed as one of the layers of a multi-layered P2P architecture. In this context, each peer uses a local clustering algorithm to cluster its resources and uses the clusters' centroids as its description of interests. Message walks are then used to create clusters of similar peers and user queries are forwarded to nodes within a cluster. Similarly the creation of groups of peers using the probability to answer the same set of queries is studied in [158], while in [120] peer clustering is achieved by comparing Bloom filter summaries of XML documents, and the clustering information is used for path query answering.

2.3.2 Information Retrieval in Super-Peer Networks

Another class of P2P systems over which IR functionality has been developed is super-peer systems discussed in Section 2.1.2. Here we survey some interesting approaches.

Keyword Search

In super-peer networks, initial work on keyword search was proposed by modifying Sun's JXTA Framework. In JXTA Search [206], an extension of the current JXTA framework that accommodates distributed information routing, is presented. A JXTA Search network is a hierarchical P2P network that consists of super-peers responsible for query routing, and clients that are either information providers or information consumers. In JXTA Search, information providers publish their content in the form of query descriptions they are capable (and willing) to answer, while information consumers submit queries to the network and these queries are routed appropriately to all interested providers.

Vector Space Model

Digital library scenarios were the dominant application used to develop hierarchical full-text query services. In [131] the authors study the problem of content-based retrieval in distributed digital libraries focusing on resource selection and document retrieval. They propose to use a 2-level hierarchical P2P network where digital libraries (called *leaf nodes*) are clients that cluster around directory nodes that form an unstructured P2P network in the second level of the hierarchy. In the same spirit, [168] investigates robustness issues in networks where directory replication but no data replication are available, while [132] examines result merging algorithms that can be utilised by hubs. In a more recent paper [130] the authors define the concept of neighborhood in hierarchical P2P networks and use this concept to devise a method for super-peer selection and ranking. In a similar fashion, [114, 115] propose an architecture for IR based clustering of peers in semi-collaborating overlay networks.

Top-k Query Processing

A strand of work that represents a more qualitative approach to the retrieval of relevant documents over structured overlays is taken by considering top-k retrieval and PageRank utilisation in a P2P setting. Since users are generally not interested in large answer sets, and generally prefer a small ordered set of the most relevant answers to a query, [29, 150] proposes a super-peer network especially designed for decentralised top-k retrieval. This functionality is made possible through the use of local rankings and a rank merging algorithm. Finally, [55] proposes a Personalised PageRank in a P2P setting and uses this algorithm to improve the selection of neighbors in neighbor-based searches.

2.3.3 Information Retrieval in Structured Peer-to-Peer Networks

[95] was one of the early papers setting the research agenda for P2P complex query processing on top of DHTs. The authors discuss the ability to build information retrieval functionality over text databases on top of structured overlays, while iden-

tifying the need for a DHT-agnostic API that will facilitate the portability of applications built on top of such overlays. In a similar spirit, [127] discusses the feasibility of Web search in a P2P environment and estimates the difficulty of the problem. The authors revisit known techniques and optimisations from different research areas and apply them to the DHT setting. They conclude that, obviously, naive implementations of Web search are not to be considered effective, and that a combination of optimisations is necessary.

Keyword Search

Supporting keyword search in a structured overlay did not receive much attention from the research community. The reasons for that were that such functionality was straightforward to implement on top of a DHT, and that researchers started looking into more sophisticated data models and query languages trying to harness the advantages of DHTs. A straightforward approach to support keyword searching in P2P networks is presented in [171]. Each node in the network is responsible for a specific keyword through the DHT hash function. The authors focus on multiple keyword queries and use a combination of optimisation techniques including Bloom filters, virtual hosts, caching and incremental results to reduce network traffic and balance the load among the peers.

Vector Space Model

A significant number of approaches tried to support VSM on top of structured overlays. Meteorograph [100] was one of the early papers to deal with the problem of similarity search over structured P2P networks. The authors devise a home-brewed structured overlay network, called Tornado [101], and describe how to support similarity search and ranked search in a linear hash addressing space overlay. In this work the authors also discuss the load balancing issues associated with such an approach and propose initial solutions. Another approach focusing more on architectural issues of building a P2P search engine was ODISSEA [184]. In ODISSEA a two tier architecture is adopted, where the lower tier nodes of the system are implemented on top of Pastry DHT. These nodes provide the search middleware

that is responsible for bringing together the upper tier nodes, namely update clients and query clients. This system was one of the first attempts to utilise a DHT in a super-peer environment.

While most of related papers utilise a DHT to route the queries to appropriate peers, Minerva [35] follows a different approach. In Minerva the structured overlay offers a conceptually global, but physically distributed directory, that maintains IR-style *statistics* and *quality of service information*. This information is exploited by querying peers, and most relevant ones according to database selection algorithms [33] are contacted. Emphasis is on peer autonomy and several extensions have been proposed. These extensions take into account overlap between peers' documents [34, 142] to improve database selection, and provide algorithms for the distribution of data and processing to cope with the problem of load balancing [141].

Latent Semantic Indexing

Since much of research was focused on the VSM model, other approaches that coped with the rather popular Latent Semantic Indexing (LSI) model appeared. In these efforts, researchers focused on extracting feature vectors from documents to allow the retrieval of relevant documents even though some of the query keywords do not exist in them. This approach has been followed to alleviate problems with polysemy[11] and synonymity[12] observed with exact keyword matching. pSearch [188, 189] was the first P2P system that used LSI to reduce the feature vectors of the documents. In pSearch the authors propose the usage of a multi-dimensional CAN to efficiently distribute document indices in the P2P network. In [175] a similar approach is proposed, and Chord DHT is used to index the documents and route the queries to appropriate peers. In this work the authors claim that scalability issues of pSearch are improved, since their approach is independent of corpus size, while all types of data (documents, images, music files etc.) can be queried, given that a meaningful feature extraction method for this type of data exists.

[11] Word with Greek roots, meaning that a single word has multiple meanings
[12] Word with Greek roots, meaning that multiple words have the same meaning

Distributed PageRank

Ranking of a page based on link structure and hub/authority computation is an important technique that has boosted search engines' effectiveness. Two prominent examples of determining the "importance" of a page are HITS [116] and Google's PageRank algorithm [39]. The need for such techniques in a P2P setting has driven work in this area. Early work presented in [2] has identified the problem of centralised approaches to ranking and presented a ranking algebra as a formal framework for ranking computation. They show that the use of only local information can approximate PageRank that is based on global information, making it an appealing approach for a P2P-based search engine. Moreover, [176] presents a distributed implementation of the PageRank algorithm in a P2P environment that also supports incremental computation of PageRank values as new documents are inserted into the system. Then the authors integrate their algorithm with existing P2P search algorithms for both structured and unstructured networks, and propose an incremental search algorithm that uses their distributed PageRank in a DHT environment. The same year with the work presented above, another approach [178] to distributed PageRank computation was presented in a different forum. The authors use structured overlays as the routing infrastructure to support distributed page ranking and introduce a method for indirect transmission to reduce communication overhead between the different page "rankers". Their approach is reported to converge to the ranks computed by the centralised counterpart, while convergence time depends on network characteristics.

Top-k Query Processing

Important qualitative techniques such as top-k query processing could not avoid receiving attention from the research community. In an elegant approach to the problem, a framework coined KLEE [140], designed to support distributed top-k algorithms for wide-area networks, is put forward. In this framework the authors describe how to utilise a fixed number of communication phases to achieve smaller response times in a distributed environment. One of the main contributions of the

KLEE approach is the flexibility to trade-off execution cost and result quality by using approximation techniques in a P2P setting.

Other Approaches

The usage of DHTs as the routing substrate for P2P IR has also been introduced to the domain of digital libraries. Recent proposals include OverCite [183] and [28]. OverCite proposes a distributed alternative for the scientific literature digital library CiteSeer[13], using a DHT infrastructure to harness distributed resources (storage, computational power, etc.). The claim behind OverCite is that due to resource limitations of CiteSeer a wide variety of useful features are not supported. Paper [28] argues that IR techniques need collection-wide information and proposes an indexing scheme that deals with the storage, the distribution and the computation of such information in federated library collections.

Architectural issues for P2P information retrieval are discussed in [5], where the authors develop a generic architecture of such a system to favor reusability of system components and (eventually) interoperability of different solutions. The architecture of such a system is decomposed into four components and each layer uses operations of the layer(s) underneath to perform its own operations.

Most of the work on P2P IR has primarily concentrated on the efficient distribution of the index to the peers and on result quality, with the goal of eventually distributing Google [91]. Global statistics such as document frequency are assumed to be available in most of the settings. [117] studies the feasibility of collecting and maintaining such statistics in a DHT environment and introduces optimisation techniques to improve the performance of the overlay network.

2.4 Information Filtering

Information Retrieval and Information Filtering (or selective dissemination of information or publish/subscribe) are often referred as two sides of the same coin [30]. Although many of the underlying issues are similar in retrieval and filtering, since

[13] `http://citeseer.ist.psu.edu`

in both cases a document needs to be matched against an information need, the design issues, the techniques and algorithms devised to increase filtering efficiency differ significantly. In this section we present initial work on IF from the IR community and summarise important approaches from database and distributed systems researchers. Finally, we focus on research conducted in IF for P2P networks.

2.4.1 Information Filtering in Information Retrieval, Databases and Distributed Systems

Historically, work on selective dissemination of information started by a 1958 article of Luhn [133], where a "Business Intelligence System" is described. In his concept, individual users would have their interests described in profiles, and a text selection system would produce lists of new documents that would allow users to choose between ordering a new document or not. At that day, the selection module was described using the terms *selective dissemination of new information*. The term *information filtering* was coined later by Denning in [71], where he described the need to filter incoming mail messages to sort them in order of urgency. Here we will discuss only the papers that are more relevant to our work and mainly those referring to content filtering (now commonly referred as *content-based filtering*).

Early approaches to information filtering by IR researchers focused mainly on appropriate representations of user interests [145] and on improving filtering effectiveness [103]. In [145] behaviour monitoring and a substring indexing method is proposed in order to decide which documents are of interest to the user. In [103] filtering is addressed using ensemble methods from machine learning, where combinations of strategies are explored as a means to increase filtering effectiveness. Other approaches include statistical filtering systems such as LSI-SDI [80], that uses the LSI method to filter incoming documents.

One of the first papers in this area to address performance is [31], where an information filtering system capable of scaling up to large filtering tasks is described. The authors assume a server that receives documents at a high rate, and propose algorithms that support vector space queries by improving the algorithm SQI of

[210]. InRoute [41] was another influential system based on inference networks with emphasis on filtering efficiency. InRoute creates documents and query networks and uses belief propagation techniques to filter incoming documents. Other works in the area mainly focus on adaptive filtering [40, 220] and how vector space queries and their dissemination thresholds are adapted based on documents processed in the past.

Apart from the statistical filtering approaches described above, filtering systems based on the Boolean model have also been developed. A representative example is LMDS [217], that uses least frequent trigrams to allow for fast processing of incoming documents. In LMDS, profiles are indexed under the least frequent trigram, whereas documents are represented as a sequence of trigrams. At filtering time a table lookup determines which profiles match the incoming document. Since false positives may occur, a second stage is necessary to determine the actual matches.

News filtering [61, 86] is also an active research area closely related to IF. In this context however the focus is on in personalisation, duplicate elimination (or information novelty) and freshness of the results shown to the user, whereas in our context we emphasise information quality, scalability and efficiency.

Most of the work on information filtering in the database literature has its origins in the paper [81] which also used the term selective dissemination of information. Their preliminary work on the system DBIS appears in [19]. The term publish/subscribe system, which comes from distributed systems, has also been used in this context by database researchers. Another influential system is SIFT [209, 211] where publications are documents in free text form and queries are conjunctions of keywords. SIFT was the first system to emphasize query indexing as a means to achieve scalability in pub/sub systems [209]. Later on, similar work concentrated on pub/sub systems with data models based on attribute-value pairs and query languages based on attributes with arithmetic and string comparison operators (e.g., Le Subscribe [75], the monitoring subsystem of Xyleme [151] and others). [43] is also notable because it considers a data model based on attribute-value pairs but goes beyond conjunctive queries – the standard class of queries considered by other systems [75]. Subscription summarisation to support pub/sub functionality was also

a prominent example of the pre-DHT era, with works such as [193]. In the context of XML data management [121], recent work has concentrated on publications that are XML documents and queries that are subsets of XPath or XQuery (e.g., XFilter [18], YFilter [73], Xtrie [48] and *xmltk* [92]). All these papers discuss sophisticated filtering algorithms based on indexing queries.

In the area of distributed systems and networks various pub/sub systems have been developed over the years. Researchers have utilized here various data models based on channels, topics and attribute-value pairs (exactly like the models of the database papers discussed above) [45]. The latter systems are called content-based like in the IR literature, as attribute-value data models are flexible enough to express the content of messages in various applications. The query languages of these systems are based on Boolean combinations of arithmetic and string operations. Work in this area has concentrated not only on filtering algorithms as in the database papers surveyed above, but also on distributed pub/sub architectures [12, 45]. SIENA [45] is probably the most elegant example of a system to be developed in this area. SIENA uses a data model and language based on attribute-value pairs and demonstrates how to express notifications, subscriptions and advertisements in this language.

2.4.2 Information Filtering in Peer-to-Peer Networks

The core ideas of SIENA have recently been used by our group in the pub/sub systems DIAS [123] and P2P-DIET [104, 125, 154]. In some sense, the approach of DIAS and P2P-DIET puts together prominent ideas from the database and distributed systems tradition in a single unifying framework. Another important contribution of P2P-DIET is that it demonstrates how to support, by very similar protocols, the traditional *ad-hoc* or *one-time* query scenarios of typical super-peer systems [212] and the pub/sub features of SIENA [45].

With the advent of distributed hash-tables (DHTs) such as CAN, Chord and Pastry, a new wave of pub/sub systems based on DHTs has appeared. Scribe [174] is a topic-based publish/subscribe system based on Pastry. Hermes [160] is similar

to Scribe because it uses the same underlying DHT (Pastry) but it allows more expressive subscriptions by supporting the notion of an event type with attributes. Each event type in Hermes is managed by an event broker which is a rendezvous node for subscriptions and publications related to this event. Related ideas appear in [186] and [190]. PeerCQ [90] is another notable pub/sub system implemented on top of a DHT infrastructure. The most important contribution of PeerCQ is that it takes into account peer heterogeneity and extends consistent hashing [112] with simple load balancing techniques based on appropriate assignment of peer identifiers to network nodes.

Meghdoot [93] is a recent pub/sub system implemented on top of a CAN-like DHT. Meghdoot supports an attribute-value data model and offers new ideas for the processing of subscriptions with range predicates (e.g., the price is between 20 and 40 Euros) and load balancing. A P2P system with an attribute-value data model similar to Meghdoot (utilized in the implementation of a pub/sub system for network games) is Mercury [37, 38]. Supporting a rich set of queries in the context of a pub/sub system is also the target of PastryStrings [11], that utilises prefix-based routing to facilitate both numerical and string queries.

New approaches that use a DHT as the routing infrastructure to build filtering functionality for more IR-based models and languages have recently been introduced. pFilter [187] is the closest system to the ideas presented in this thesis. It uses a hierarchical extension of CAN [165] to filter unstructured documents and relies on multi-cast trees to notify subscribers. VSM and LSI can be used to match documents to user queries. By comparing pFilter with the proposals of this thesis, we can see that we have a more expressive data model and query language, and do not need to maintain multi-cast trees to notify subscribers. However, the multi-cast trees of pFilter take into account network distance something that we do not consider at all in this thesis. We also consider load balancing issues that are not studied in pFilter. Finally, regarding routing, it would be interesting to compare experimentally a hierarchical extension of our work with the hierarchical routing protocols of pFilter. Finally, supporting prefix and suffix queries in string attributes is the focus of the DHTStrings system [10], which utilises a DHT-agnostic

architecture to develop algorithms for efficient multi-dimensional event processing.

2.5 Conclusions

In this chapter, related literature on P2P systems was surveyed, starting with three pioneer systems that popularised the P2P paradigm, and concluding with an extensive survey of the state of the art in structured overlays. Subsequently, we reviewed work carried out at the intersection of information retrieval/filtering and P2P systems.

In the next chapters we present our contributions to each one of the technical challenges posed in Section 1.1. The next chapter concentrates on presenting three progressively more expressive data models and query languages designed using concepts from IR, and focuses on model-theoretic questions for the logics of those models.

Chapter 3

Data Models and Query Languages Based on Information Retrieval

In this chapter we study three well-known data models of IR [24] and digital libraries [49–51], which we have called $\mathcal{WP}$, $\mathcal{AWP}$ and $\mathcal{AWPS}$ in [123–125, 200, 202, 203]. Variations of these models are in use in most deployed digital libraries, text extenders for relational database systems, search engines and P2P systems for information retrieval and filtering, and have also been used for annotating and querying resources in the context of this work. Chapters 4 and 5 will show how to process queries expressed in $\mathcal{AWPS}$ on top of DHTs, while Chapter 6 presents the local data structures used for indexing $\mathcal{AWP}$ queries. The main contribution of the chapter is the study of the complexity of query satisfiability and entailment for models $\mathcal{WP}$ and $\mathcal{AWP}$ using techniques from propositional logic and computational complexity. The results of this chapter have been published in [126].

Data model $\mathcal{WP}$ is based on *free text* and its query language is based on the Boolean model for *word patterns*. Word patterns are formulas that enable the expression of constraints on the existence, non-existence or proximity of words in a text document. Data model $\mathcal{AWP}$ extends $\mathcal{WP}$ with *named attributes* with free text as values. The query language of $\mathcal{AWP}$ is also a simple extension of the query

language of $\mathcal{WP}$ so that attributes are included. Finally, the model $\mathcal{AWPS}$ extends $\mathcal{AWP}$ by introducing a "similarity" operator in the style of modern IR, by resorting to well-known IR tools such as VSM or LSI [24].

Models such as $\mathcal{WP}$ that are based on word patterns were introduced in the early days of IR and have been implemented in many digital library systems in wide use today [24]. Word patterns are also used in (a) all current search engines, (b) advanced IR models such as the model of proximal nodes [148] which allows proximity operators between arbitrary structural components of a document (e.g., paragraphs or sections), and (c) recent full-text extensions to XML-based languages e.g., TeXQuery [20].

The model $\mathcal{AWP}$ has been recently used in our systems DIAS and P2P-DIET [106, 123, 200]. DIAS [123] is a distributed alert service for digital libraries which utilises a P2P architecture and protocols similar to that of the event dissemination system SIENA [44]. It uses $\mathcal{WP}$ and $\mathcal{AWP}$ as an expressive data model and query language for textual information. P2P-DIET [106] is the ancestor of DIAS and uses $\mathcal{AWP}$ as a metadata model for describing and querying digital resources. Centralised filtering algorithms especially designed for model $\mathcal{AWP}$ are discussed in detail in Chapter 6. Finally, $\mathcal{AWPS}$ is used in the DHT-based P2P systems DHTrie, discussed in Chapter 5 and [203], and LibraRing, presented in Chapter 4 and [202].

In the database literature, word patterns have been studied by Chang and colleagues in the context of integrating heterogeneous digital libraries [49–51]. The model $\mathcal{AWP}$ is essentially the model of [50] but with a slightly different class of word patterns. Text extensions of commercial relational database products (e.g., Oracle 10g) also offer full support for word patterns.

Even though many deployed systems are using $\mathcal{WP}$ and $\mathcal{AWP}$ and many papers have appeared on their variations, only [49–51, 123, 124] have studied in depth the *logical* foundations of these data models. As we have previously discussed in [124], we would like to develop information retrieval and filtering systems in a *principled* and *formal* way. With this motivation and the architectures of [106, 123, 200, 202, 203] in mind, we have posed the following requirements for models and languages to be

used in information retrieval and filtering systems [124]:

1. *Expressiveness.* The languages for documents and queries must be rich enough to satisfy the demands of information consumers and the capabilities of information providers.

2. *Formality.* The syntax and semantics of the proposed models and languages must be defined formally.

3. *Computational efficiency.* The following problems should be defined formally and algorithms must be provided for their efficient solution (keeping in mind that there will be a trade-off with the expressiveness requirement):

   (a) The *satisfiability* problem: Deciding whether a query can be satisfied by any document at all.

   (b) The *satisfaction* problem: Deciding whether a document satisfies a query.

   (c) The *retrieval* problem: Given a collection of documents D and an incoming query q, find all documents $d \in D$ that satisfy q.

   (d) The *filtering* problem: Given a collection of queries Q and an incoming document d, find all queries $q \in Q$ that satisfy d.

   (e) The *entailment* problem: Deciding whether a query is more or less "general" than another.

In this work we define formally the models $\mathcal{WP}$, $\mathcal{AWP}$ and $\mathcal{AWPS}$ and concentrate on *model-theoretic questions* for the logics of $\mathcal{WP}$ and $\mathcal{AWP}$ that have been ignored so far. Additionally in the context of this thesis, we provide centralised and distributed algorithms for the filtering problem (see Chapters 6 and 5 respectively and also papers [200, 203]). We study the model theory of $\mathcal{WP}$ and $\mathcal{AWP}$ and especially questions related to satisfiability and entailment. We show that the satisfiability problem for queries in $\mathcal{WP}$ and $\mathcal{AWP}$ is $\mathcal{NP}$-complete and the entailment problem is co$\mathcal{NP}$-complete. We also discuss cases where these problems can be solved in polynomial time. Our results are original and complement the studies of [50, 124] where no such complexity questions were posed.

In the next sections we introduce the models $\mathcal{WP}$, $\mathcal{AWP}$ and $\mathcal{AWPS}$ and explain our formal data modeling perspective. Sections 3.3 and 3.4 present our complexity results on satisfiability and entailment, while Section 3.5 briefly presents other data models (and systems) related to the ones studied here. The last section summarises the results presented and concludes the chapter.

3.1 The Models $\mathcal{WP}$ and $\mathcal{AWP}$

Let us start by presenting the data model $\mathcal{WP}$ and its query language. $\mathcal{WP}$ has been inspired by [49]. It assumes that textual information is in the form of *free text* and can be queried by *word patterns* (hence the acronym for the model).

We assume the existence of a finite *alphabet* $\mathbf{\Sigma}$. A *word* is a finite non-empty sequence of letters from $\mathbf{\Sigma}$. We also assume the existence of a (finite or infinite) set of words called the *vocabulary* and denoted by $\mathcal{V}$. A *text value* s of length n over vocabulary $\mathcal{V}$ is a total function $s : \{1, 2, \ldots, n\} \rightarrow \mathcal{V}$. In other words, a text value s is a finite sequence of words from the assumed vocabulary and $s(i)$ gives the i-th element of s. $|\, s \,|$ will denote the length of text value s (i.e., the number of words in it).

We now give the definition of word pattern. We assume the existence of a set of *(distance) intervals* $\mathcal{I} = \{[l, u] : \ l, u \in \mathbb{N}, l \geq 0 \text{ and } l \leq u\} \cup \{[l, \infty) : \ l \in \mathbb{N} \text{ and } l \geq 0\}$. Let $\boldsymbol{i}$ be an interval in $\mathcal{I}$. We will denote the left-endpoint (respectively right-endpoint) of $\boldsymbol{i}$ by $inf(\boldsymbol{i})$ (respectively $sup(\boldsymbol{i})$).

Definition 1 *Let $\mathcal{V}$ be a vocabulary. A* word pattern *over vocabulary $\mathcal{V}$ is a formula in any of the following forms:*

1. *w, where w is a word of $\mathcal{V}$.*
2. *$w_1 \prec_{\boldsymbol{i}_1} \cdots \prec_{\boldsymbol{i}_{n-1}} w_n$, where $w_1, \ldots, w_n$ are words of $\mathcal{V}$ and $\boldsymbol{i}_1, \ldots, \boldsymbol{i}_{n-1}$ are intervals of $\mathcal{I}$.*
3. *$\neg\phi$, $\phi_1 \vee \phi_2$ or $\phi_1 \wedge \phi_2$, where ϕ, ϕ_1 and ϕ_2 are word patterns.*

Example 1 *The following are word patterns:*

$$constraint \wedge (optimisation \vee programming)$$
$$\neg algorithms \wedge ((complexity \prec_{[1,5]} satisfaction) \vee (complexity \prec_{[1,8]} filtering))$$

Operators $\prec_i$ are called *proximity operators* and are generalizations of the traditional IR operators kW and kN [49]. Proximity operators are used to capture the concepts of *order* and *distance* between words in a text document. They can be used to construct formulas of $\mathcal{WP}$ that we will call *proximity word patterns* (Case 2 of Definition 1). The proximity word pattern $w_1 \prec_{[l,u]} w_2$ stands for "word w_1 is *before* w_2 and is separated by w_2 by *at least* l and *at most* u *words*". The interpretation of proximity word patterns with more than one operator $\prec_i$ is similar.

Traditional IR systems have proximity operators kW and kN where k is a natural number. The proximity word pattern $wp_1\ kW\ wp_2$ stands for "word pattern wp_1 is before wp_2 and is separated by wp_2 by at most k words". In our work this can be captured by $wp_1 \prec_{[0,k]} wp_2$. The operator kN is used to denote distance of at most k words where the order of the involved patterns does not matter. In $\mathcal{WP}$ the expression $wp_1\ kN\ wp_2$ can be approximated by $wp_1 \prec_{[0,k]} wp_2 \vee wp_2 \prec_{[0,k]} wp_1$. [49] gives an example (page 23) that demonstrates why these two expressions are not equivalent given the meaning of operator kN. The example involves a text value and word patterns with overlapping positions in that text value hence the difference.

The development of proximity word patterns in [49–51] follows closely the IR tradition, i.e., operators kW and kN (already mentioned above) are used together with the boolean operators AND and OR. These operators can be intermixed in arbitrary ways (e.g., $((w_1\ AND\ (w_2\ (8W)\ w_3))\ \ (10W)\ w_4)$ where w_1, w_2, w_3, w_4 are words is a legal expression), and the result of their evaluation on document databases is defined in an algebraic way. $\mathcal{WP}$ opts for an approach which is more in the spirit of Boolean logic, allows negation and carefully distinguishes word patterns with and without proximity operators. This leads to a simpler language because cumbersome (and not especially useful) constructions such as the above are avoided. In the spirit of Boolean logic, an atomic word pattern (i.e., a word or a proximity word pattern) allows us to distinguish between text values: those that satisfy it, and those that do not. Boolean operators are then given their standard semantics.

In addition to the above operators, $\mathcal{WP}$ allows the expression of simple order constraints between words using operators $\prec_{[0,\infty]}$. Order constraints of the form $\prec_{[0,\infty]}$ between various text structures are also present in more advanced text model proposals such as the model of proximal nodes of [148].

Definition 2 *A word pattern will be called* positive *if it does not contain negation. A word pattern will be called* proximity-free *if it does not contain formulas of the form* $w_1 \prec_{\boldsymbol{i}_1} \cdots \prec_{\boldsymbol{i}_{n-1}} w_n$. *A word pattern will be called* conjunctive *if it does not contain disjunction and negation.*

Example 2 *The following are positive word patterns:*

$$satisfiability,$$
$$local \wedge search \wedge algorithms,$$
$$information \wedge (retrieval \vee dissemination),$$
$$logic \prec_{[0,1]} computational \prec_{[0,0]} complexity$$

The first three are proximity-free word patterns. The first, second and fourth word pattern is conjunctive.

Definition 3 *Let* $\mathcal{V}$ *be a vocabulary,* s *a text value over* $\mathcal{V}$ *and* wp *a word pattern over* $\mathcal{V}$. *The concept of* s satisfying wp *(denoted by* $s \models wp$*) is defined as follows:*

1. *If* wp *is a word of* $\mathcal{V}$ *then* $s \models wp$ *iff there exists* $p \in \{1, \ldots, |s|\}$ *such that* $s(p) = wp$.

2. *If* wp *is a proximity word pattern of the form* $w_1 \prec_{\boldsymbol{i}_1} \cdots \prec_{\boldsymbol{i}_{n-1}} w_n$ *then* $s \models wp$ *iff there exist* $p_1, \ldots, p_n \in \{1, \ldots, |s|\}$ *such that, for all* $j = 2, \ldots, n$ *we have* $s(p_j) = w_j$ *and* $p_j - p_{j-1} - 1 \in \boldsymbol{i}_{j-1}$.

3. *If* wp *is of the form* $\neg wp_1, wp_1 \wedge wp_2, wp_1 \vee wp_2$ *or* (wp_1) *then* $s \models wp$ *is defined exactly as satisfaction for Boolean logic.*

A word pattern wp *is called* satisfiable *if there is a text value* s *that satisfies it. Otherwise, it is called* unsatisfiable.

Example 3 *The word patterns of Examples 1 and 2 are satisfiable. Word patterns*

$$\neg programming \wedge (constraint \prec_{[0,0]} programming),$$
$$(constraint \prec_{[0,0]} programming) \wedge \neg(constraint \prec_{[0,2]} programming)$$

are unsatisfiable.

Definition 4 *Let wp_1 and wp_2 be word patterns. We will say that wp_1* entails *wp_2 (denoted by $wp_1 \models wp_2$) iff for every text value s such that $s \models wp_1$, we have $s \models wp_2$. If $wp_1 \models wp_2$ and $wp_2 \models wp_1$ then wp_1 and wp_2 are called* equivalent *(denoted by $wp_1 \equiv wp_2$).*

Example 4 *Word pattern constraint $\wedge$ programming entails word pattern constraint. Word pattern*

$$optimization \wedge (constraint \prec_{[0,0]} programming)$$

entails constraint $\prec_{[0,10]}$ programming.
Finally, word patterns

$$constraint \prec_{[0,4]} programming,$$
$$constraint \wedge (constraint \prec_{[0,4]} programming)$$

are equivalent.

Proposition 3.1.1 *Let wp_1 and wp_2 be two word patterns. $wp_1 \models wp_2$ iff $wp_1 \wedge \neg wp_2$ is unsatisfiable.*

Let us close this section by pointing out that proximity word patterns have been considered as atomic formulas of $\mathcal{WP}$ (Definition 1) because, in general, negation cannot be moved inside a proximity word pattern as in the case of Boolean operators. The interested reader can be persuaded by trying to do this for the following formula:

$$\neg(luxurious \prec_{[0,3]} hotel \prec_{[0,3]} beach)$$

If we restrict our attention to proximity formulas with a single proximity operator, this restriction can easily be lifted. For example, the word pattern

$$\neg(luxurious \prec_{[0,3]} hotel)$$

is equivalent to the following:

$$\neg luxurious \vee \neg hotel \vee hotel \prec_{[0,\infty]} luxurious \vee luxurious \prec_{[4,\infty]} hotel$$

Let us now use the machinery of $\mathcal{WP}$ to define data model $\mathcal{AWP}$. The new concept of $\mathcal{AWP}$ is the concept of *attribute* with value free text (in the acronym $\mathcal{AWP}$, the letter $\mathcal{A}$ stands for "attribute").

We assume the existence of a countably infinite set of attributes $\mathbf{U}$ called the *attribute universe*. A *document schema* $\mathcal{D}$ is a pair $(\mathcal{A}, \mathcal{V})$ where $\mathcal{A}$ is a subset of the attribute universe $\mathbf{U}$ and $\mathcal{V}$ is a vocabulary. A *document* d over schema $(\mathcal{A}, \mathcal{V})$ is a set of attribute-value pairs (A, s) where $A \in \mathcal{A}$, s is a text value over $\mathcal{V}$, and there is at most one pair (A, s) for each attribute $A \in \mathcal{A}$.

Example 5 *The following is a document over schema* $(\{AUTHOR, TITLE, ABSTRACT\}, \mathcal{V})$*:*

$$\{\ (AUTHOR, \text{“John Brown”}),$$
$$(TITLE, \text{“Local search and constraint programming”}),$$
$$(ABSTRACT, \text{“In this paper we show that ...”})\ \}$$

The syntax of the query language of $\mathcal{AWP}$ is given by the following recursive definition.

Definition 5 *A* query *over schema* $(\mathcal{A}, \mathcal{V})$ *is a formula in any of the following forms:*

1. $A \sqsupseteq wp$*, where* $A \in \mathcal{A}$ *and* wp *is a word pattern over* $\mathcal{V}$ *(this is read as "A contains word pattern* wp*").*

2. $A = s$*, where* $A \in \mathcal{A}$ *and* s *is a text value over* $\mathcal{V}$*.*

3. $\neg\phi,\ \phi_1 \vee \phi_2,\ \phi_1 \wedge \phi_2$, *where* ϕ, ϕ_1 *and* ϕ_2 *are queries.*

Example 6 *The following is a query over the schema shown in Example 5:*

$$AUTHOR \sqsupseteq Brown \quad \wedge$$
$$TITLE \sqsupseteq search \wedge (constraint \prec_{[0,0]} programming)$$

Definition 6 *Let* $\mathcal{D}$ *be a document schema,* d *a document over* $\mathcal{D}$ *and* ϕ *a query over* $\mathcal{D}$*. The concept of document* d satisfying *query* ϕ *(denoted by* $d \models \phi$*) is defined as follows:*

1. *If* ϕ *is of the form* $A \sqsupseteq wp$ *then* $d \models \phi$ *iff there exists a pair* $(A, s) \in d$ *and* $s \models wp$*.*

2. *If* ϕ *is of the form* $A = s$ *then* $d \models \phi$ *iff there exists a pair* $(A, s) \in d$*.*

3. *If* ϕ *is of the form* $\neg\phi_1$ *then* $d \models \phi$ *iff* $d \not\models \phi_1$*. Similarly for* $\wedge$ *and* $\vee$*.*

Example 7 *The query of Example 6 is satisfied by the document of Example 5.*

Proposition 3.1.2 *Let* A *be an attribute and* wp_1, wp_2 *be word patterns. Then, the following equivalences hold:*

1. $\neg A \sqsupseteq wp \equiv A \sqsupseteq \neg wp$

2. $A \sqsupseteq (wp_1 \wedge wp_2) \equiv (A \sqsupseteq wp_1) \wedge (A \sqsupseteq wp_2)$

3. $A \sqsupseteq (wp_1 \vee wp_2) \equiv (A \sqsupseteq wp_1) \vee (A \sqsupseteq wp_2)$

4. $\neg(A \sqsupseteq (wp_1 \wedge wp_2)) \equiv (\neg A \sqsupseteq wp_1) \vee (\neg A \sqsupseteq wp_2)$

5. $\neg(A \sqsupseteq (wp_1 \vee wp_2)) \equiv (\neg A \sqsupseteq wp_1) \wedge (\neg A \sqsupseteq wp_2)$

Definition 7 *A query is called* atomic *if it is of the form* $A = t$ *where* t *is a text value, or* $A \sqsupseteq wp$ *where* wp *is a word or a proximity word pattern. A query is called* conjunctive *if it does not contain disjunction.*

Example 8 *The following queries are atomic:*

$$AUTHOR = \text{“}James\ Brown\text{”},\ TITLE \sqsupseteq search,$$

$$ABSTRACT \sqsupseteq constraint \prec_{[0,0]} programming$$

Proposition 3.1.3 *Every query is equivalent to a Boolean combination of atomic queries.*

Proof: Use the first three equivalences of Proposition 3.1.2 repeatedly. ■

3.2 Extending $\mathcal{AWP}$ with Similarity

Let us now define our third data model $\mathcal{AWPS}$ and its query language. $\mathcal{AWPS}$ extends $\mathcal{AWP}$ with the concept of *similarity* between two text values (the letter $\mathcal{S}$ stands for similarity). The idea here is to have a “soft” alternative to the “hard” operator $\sqsupseteq$. This operator is very useful for queries such as “I am interested in papers that have the term local search in their title” which can be written in $\mathcal{AWP}$ as

$$TITLE \sqsupseteq (local \prec_{[0,0]} search)$$

but it might not be very useful for queries “I am interested in papers about the use of local search techniques for the problem of test pattern optimisation”.

The desired functionality can be achieved by resorting to an important tool of modern IR: the *weight* of a word as defined in the Vector Space Model (VSM) [24, 137, 207]. In VSM, documents (text values in our terminology) are conceptually represented as vectors. If our vocabulary consists of n distinct words then a text value s is represented as an n-dimensional vector of the form $(\omega_1, \ldots, \omega_n)$ where ω_i is the weight of the i-th word (the weight assigned to a non-existent word is 0). With a good weighting scheme, the VSM representation of a document can be a surprisingly good model of its semantic content in the sense that “similar” documents have very close semantic content. This has been demonstrated by many successful IR systems

[24] or database systems adopting ideas from IR (see for example, WHIRL [58])[1].

In VSM, the weight of a word is computed using the heuristic of assigning higher *weights* to words that are frequent in a document and *infrequent* in the collection of documents available. This heuristic is made concrete using the concepts of word frequency and the inverse document frequency defined below.

Definition 8 *Let w_i be a word in document d_j of a collection C. The* term frequency *of w_i in d_j (denoted by tf_{ij}) is equal to the number of occurrences of word w_i in d_j. The* document frequency *of word w_i in the collection C (denoted by df_i) is equal to the number of documents in C that contain w_i. The* inverse document frequency *of w_i is then given by $idf_i = \frac{1}{df_i}$. Finally, the number $tf_{ij} \cdot idf_i$ will be called the* weight *of word w_i in document d_j and will be denoted by ω_{ij}.*

At this point we should stress that the concept of inverse document frequency assumes that there is a *collection* of documents which is used in the calculation. In Chapters 4, 5 and 6 we assume that for each attribute A there is a collection of text values C_A that is used for calculating the *idf* values to be used in similarity computations involving attribute A (the details are given below). C_A can be a collection of recently processed text values as suggested in [76, 211].

We are now ready to define the main new concept in $\mathcal{AWPS}$, the similarity of two text values. The similarity of two text values s_q and s_d is defined as the cosine of the angle formed by their corresponding vectors[2]:

$$sim(s_q, s_d) = \frac{s_q \cdot s_d}{\|s_q\| \cdot \|s_d\|} = \frac{\sum_{i=1}^{N} w_{q_i} \cdot w_{d_i}}{\sqrt{\sum_{i=1}^{N} w_{q_i}^2 \cdot \sum_{i=1}^{N} w_{d_i}^2}} \tag{3.1}$$

[1]Sometimes in IR systems (or systems adopting ideas from IR) word *stems*, produced by some stemming algorithm [163], are forming the vocabulary instead of words. Additionally, *stopwords* (e.g., "the") are eliminated from the vocabulary. These important details have no consequence for the theoretical results of this work, but it should be understood that our approaches presented in Chapters 4, 5 and 6 utilise these standard techniques.

[2]The IR literature gives us several very closely related ways to define the notions of weight and similarity [24, 137, 207]. All of these weighting schemes come by the name of $tf \cdot idf$ weighting schemes. Generally a weighting scheme is called $tf \cdot idf$ whenever it uses word frequency in a monotonically increasing way, and document frequency in a monotonically decreasing way.

By this definition, similarity values are real numbers in the interval $[0, 1]$.

Let us now proceed to give the syntax of the query language for $\mathcal{AWPS}$. Since $\mathcal{AWPS}$ extends $\mathcal{AWP}$, a query in the new model is given by Definition 5 with one more case for atomic queries:

- $A \sim_k s$ where $A \in \mathcal{A}$, s is a text value over $\mathcal{V}$ and k is a real number in the interval $[0, 1]$.

Example 9 *The following are some queries in $\mathcal{AWPS}$ using the schema of Example 5:*

$$TITLE \sim_{0.6} \text{“}Local\ search\ techniques\ for\ constraint\ optimisation\ problems\text{”},$$
$$(AUTHOR \sqsupseteq (John \prec_{[0,2]} Smith)) \wedge$$
$$(TITLE \sim_{0.9} \text{“}Temporal\ constraint\ programming\text{”}),$$
$$TITLE \sim_{0.9} \text{“}Sequence\ alignment\ using\ dynamic\ programming\text{”}$$

We now give the semantics of our query language, by defining when a document satisfies a query. Naturally, the definition of satisfaction in $\mathcal{AWPS}$ is as in Definition 6 with one additional case for the similarity operator:

- If ϕ is of the form $\mathcal{A} \sim_k s_q$ then $d \models \phi$ iff there exists a pair $(A, s_d) \in d$ and $sim(s_q, s_d) \geq k$.

The reader should notice that the number k in a similarity predicate $A \sim_k s$ gives a *relevance threshold* that candidate text values s should exceed in order to satisfy the predicate. This notion of relevance threshold was first proposed in an information filtering setting by [80] and later on adopted by [211]. The reader is asked to contrast this situation with the typical information retrieval setting where a ranked list of documents is returned as an answer to a user query. This is not a relevant scenario in an information filtering system because very few documents (or even a single one) enter the system at a time, and the system needs to make a decision whether they will be forwarded or not to interested users.

A low similarity threshold in a predicate $A \sim_k s$ might result in many irrelevant documents satisfying a query. This not only decreases user satisfaction, but also

causes extra network traffic due to the creation of many notification messages (as discussed in detail in Chapters 4 and 5).On the other hand a high similarity threshold would result in very few documents (or even no documents at all) achieving satisfaction, which deprives user of potentially useful information. Relevance feedback techniques and recent techniques from adaptive IR [47] can be used to adjust this threshold and achieve better user satisfaction. However this lies beyond the focus of this thesis, which is the performance aspects of the retrieval and filtering problem, and thus it is not explored in any way

Example 10 *The first query of Example 9 is likely to be satisfied by the document of Example 5 (of course, we cannot say for sure until we know the idf factors so that the exact weights can be calculated). The second query is not satisfied, since the author specified in the query does not match the document's author. Moreover the third query is unlikely to be satisfied since the only common word between the query and Example 5 is the word "programming".*

3.3 Satisfiability and Entailment in $\mathcal{WP}$

An instance of the satisfiability problem for proximity-free word patterns can be considered as an instance of the satisfiability problem for Boolean logic (SAT) and vice versa (by interchanging the roles of words and Boolean variables). Thus, we have to consider only any complications that might arise due to proximity word patterns.

In what follows, we will need the binary operation of *concatenation* of two text values.

Definition 9 *Let s_1 and s_2 be text values over vocabulary $\mathcal{V}$. Then, the* concatenation *of s_1 and s_2 is a new text value denoted by $s_1 s_2$ and defined by the following:*

1. $|s_1 s_2| = |s_1| + |s_2|$
2. $s_1 s_2(x) = \begin{cases} s_1(x) \text{ for all } x \in \{1, \ldots, |s_1|\} \\ s_2(x - |s_1|) \text{ for all } x \in \{|s_1| + 1, \ldots, |s_2| + |s_1|\} \end{cases}$.

We will also need the concept of the *empty text value* which is denoted by ϵ and has the property $|\epsilon| = 0$. The following properties of concatenation are easily seen:

1. $(s_1 s_2)s_3 = s_1(s_2 s_3)$, for all text values s_1, s_2 and s_3.

2. $s\epsilon = \epsilon s = s$ for every text value s.

The associativity of concatenation allows us to write concatenations of more than two text values without using parentheses.

The following variant of the concept of satisfaction captures the notion of a set of positions in a text value containing *exactly* the words that contribute to the satisfaction of a *positive proximity-free* word pattern. This variant is used in Lemma 3.3.1 below and in Proposition 3.3.1.

Definition 10 *Let $\mathcal{V}$ be a vocabulary, s a text value over $\mathcal{V}$, wp a positive proximity-free word pattern over $\mathcal{V}$, and P a subset of $\{1, \ldots, |s|\}$. The concept of s* satisfying *wp* with set of positions *P (denoted by $s \models_P wp$) is defined as follows:*

1. *If wp is a word of $\mathcal{V}$ then $s \models_P wp$ iff there exists $x \in \{1, \ldots, |s|\}$ such that $P = \{x\}$ and $s(x) = wp$.*

2. *If wp is of the form $wp_1 \wedge wp_2$ then $s \models_P wp$ iff there exist sets of positions $P_1, P_2 \subseteq \{1, \ldots, |s|\}$ such that $s \models_{P_1} wp_1$, $s \models_{P_2} wp_2$ and $P = P_1 \cup P_2$.*

3. *If wp is of the form $wp_1 \vee wp_2$ then $s \models_P wp$ iff $s \models_P wp_1$ or $s \models_P wp_2$.*

4. *If wp is of the form (wp_1) then $s \models_P wp$ iff $s \models_P wp_1$.*

We also need the following notation. Let P be a subset of the set of natural numbers $\mathbb{N}$, and $x \in \mathbb{N}$. We will use the notation $P + x$ to denote the set of natural numbers $\{p + x : \ p \in P\}$.

Lemma 3.3.1 *Let s and s' be text values, wp be a positive proximity-free word pattern and $P \subseteq \{1, \ldots, |s|\}$. If $s \models_P wp$ then $ss' \models_P wp$ and $s's \models_{P+|s'|} wp$.*

Positive proximity-free word patterns are satisfiable as we show below.

Proposition 3.3.1 *If wp is a positive proximity-free word pattern then wp is satisfiable. In fact, there exists a text value s_0 such that*

1. $|s_0| \leq |\, wp \,| \cdot ops(wp)$, *where $ops(wp)$ is the number of operators of wp (or 1 if wp has no operators).*

2. *Every word of s_0 is a word of wp.*

3. $s_0 \models_{\{1,\ldots,|s_0|\}} wp$.

Proof: The proof is by induction on the structure of wp.

Base case: Let wp be a word $w \in \mathcal{V}$. In this case, wp is satisfiable because we can form a text value s_0 such that $s_0 \models_{\{1\}} w$ where $|s_0| = 1$ and $s_0(1) = w$. The conclusion of the lemma is now obviously satisfied.

Inductive step: Let wp be a positive proximity-free word pattern of the form $wp_1 \wedge wp_2$, and assume that the inductive hypothesis holds for wp_1 and wp_2. Then, we can form text values s_0^1 and s_0^2 such that $s_0^1 \models_{\{1,\ldots,|s_0^1|\}} wp_1$ and $s_0^2 \models_{\{1,\ldots,|s_0^2|\}} wp_2$. Then, from Lemma 3.3.1 we have

$$s_0^1 s_0^2 \models_{\{1,\ldots,|s_0^1|\}} wp_1$$

and

$$s_0^1 s_0^2 \models_{\{1,\ldots,|s_0^2|\}+|s_0^1|} wp_2.$$

Finally, from Definition 10 we have

$$s_0^1 s_0^2 \models_{\{1,\ldots,|s_0^1|,|s_0^1|+1,\ \ldots,|s_0^1|+|s_0^2|\}} wp_1 \wedge wp_2$$

as required. It is also easy to see that

$$\left|s_0^1 s_0^2\right| = \left|s_0^1\right| + \left|s_0^2\right| \leq$$

$$|\, wp_1 \,| \cdot ops(wp_1) + |\, wp_2 \,| \cdot ops(wp_2) <$$

$$[ops(wp_1) + ops(wp_2)] \cdot |\, wp \,| < ops(wp) \cdot |\, wp \,|.$$

The $\vee$ case is done similarly. ■

Obviously, proximity word patterns are also satisfiable.

Proposition 3.3.2 *Let wp be a proximity word pattern of the form $w_1 \prec_{\boldsymbol{i}_1} \cdots \prec_{\boldsymbol{i}_{n-1}} w_n$. Then, wp is satisfied by the text value $s = w_1 z_1 \cdots z_{n-1} w_n$ where z_l, $l = 1, \ldots, n-1$ are text values of the following form. If $inf(\boldsymbol{i}_l) > 0$ then z_l is formed by $inf(\boldsymbol{i}_l)$ successive occurrences of the special word # which is not contained in wp. Otherwise, if $inf(\boldsymbol{i}_l) = 0$ then z_l is the empty text value ϵ.*

Moreover, any text value satisfying a proximity word pattern is of a very special form.

Proposition 3.3.3 *Let wp be a proximity word pattern of the form $w_1 \prec_{\boldsymbol{i}_1} \cdots \prec_{\boldsymbol{i}_{n-1}} w_n$. If $s \models wp$ then s is of the form.*

$$s = \underbrace{? \cdots ?}_{i_0 \text{ times}} w_1 \underbrace{? \cdots ?}_{i_1 \text{ times}} w_2 \quad \cdots \quad w_{n-1} \underbrace{? \cdots ?}_{i_{n-1} \text{ times}} w_n \underbrace{? \cdots ?}_{i_n \text{ times}}$$

where $0 \leq i_0$, $i_1 \in \boldsymbol{i}_1$, ..., $i_{n-1} \in \boldsymbol{i}_{n-1}$, $0 \leq i_n$ and each occurrence of the symbol ? represents an arbitrary (and not necessarily the same) word.

Example 11 *Let us consider the proximity word pattern*

$$wp = constraint \prec_{[0,0]} programming \prec_{[0,\infty]} methods.$$

It is easy to verify that text value "many applications use constraint programming algorithms and methods to solve interesting problems" (a) is of the form set by Proposition 3.3.3 and (b) satisfies word pattern wp.

Finally, we show that any positive word pattern is satisfiable.

Proposition 3.3.4 *If wp is a positive word pattern then wp is satisfiable.*

Proof: We will construct a text value t such that $t \models wp$. If wp contains m proximity word patterns $\phi_1, \ldots, \phi_m$, text value t is of the form $s_0 s_1 \cdots s_m$ where:

- s_0 is a sequence formed by the juxtaposition of all words appearing in wp in any order, and
- for every $j = 1, \ldots, m$, s_j is a text value, formed as in Proposition 3.3.2, such that $s_j \models \phi_j$.

■

Lemma 3.3.2 *Let wp_1 and wp_2 be proximity word patterns of the following form:*

$$wp_1 = a_1 \prec_{i_1} \cdots \prec_{i_{n-1}} a_n \qquad \textit{and}$$

$$wp_2 = b_1 \prec_{j_1} \cdots \prec_{j_{m-1}} b_m$$

Word pattern wp_1 entails wp_2 iff the following conditions hold:

Condition 1 *Word pattern wp_2 is equal to $a_{p_1} \prec_{j_1} \cdots \prec_{j_{m-1}} a_{p_m}$, where $1 \leq p_1 < \cdots < p_m \leq n$.*

Condition 2 *For every $v = 1, \ldots, m-1$, we have:*

$$inf(\boldsymbol{j}_v) \leq inf(\boldsymbol{i}_{p_v}) + \cdots + inf(\boldsymbol{i}_{p_{v+1}}) + p_{v+1} - p_v - 1$$

$$sup(\boldsymbol{j}_v) \textit{ is } \begin{cases} \geq \begin{pmatrix} sup(\boldsymbol{i}_{p_v}) + \cdots + \\ sup(\boldsymbol{i}_{p_{v+1}}) + \\ p_{v+1} - p_v - 1 \end{pmatrix} & \begin{matrix} \textit{if all } sup(\boldsymbol{i}_{p_v}), \ldots, \\ sup(\boldsymbol{i}_{p_{v+1}}) \textit{ are} \\ \textit{different than } \infty \end{matrix} \\ \infty \quad \textit{otherwise} \end{cases}$$

Proof: The "if" case is obvious. For the "only if" part, let us assume that $wp_1 \models wp_2$ holds. We will prove that wp_2 is of the form set by the lemma. The proof is in three steps.

Step 1 (Condition 1) We will first prove that the words of wp_2 are a subset of the words in wp_1, i.e.,

$$\{b_1, \ldots, b_m\} \subseteq \{a_1, \ldots, a_n\}.$$

By contradiction, let us assume that there exists a word b_v, $1 \leq v \leq m$, of wp_2 such that $b_v \notin \{a_1, \ldots, a_n\}$. Let us now consider text value τ defined as:

$$\tau = a_1 \underbrace{\# \cdots \#}_{i_1 \text{ times}} a_2 \cdots a_{n-1} \underbrace{\# \cdots \#}_{i_{n-1} \text{ times}} a_n \tag{3.2}$$

where $\#$ is a special word which is not contained in wp_1 and wp_2 and $i_1 \in \boldsymbol{i}_1, \ldots, i_n \in \boldsymbol{i}_n$. It is easy to verify that τ satisfies wp_1 but, since τ does not include word b_v, it does not satisfies wp_2. Thus, we have $wp_1 \not\models wp_2$ which contradicts our initial assumption.

Step 2 (Condition 1) We will now prove that the words of wp_1 that appear in wp_2 actually appear in the same order as they do in wp_1, i.e., word pattern $wp_2 = a_{p_1} \prec_{\boldsymbol{j}_1} \cdots \prec_{\boldsymbol{j}_{m-1}} a_{p_m}$, where $1 \leq p_1 < \cdots < p_m \leq n$. By contradiction, let us assume that there exist two distinct words $b_v = a_{p_v}$ and $b_{v'} = a_{p_{v'}}$, $1 \leq v < v' \leq m$, of wp_2 such that $p_v \geq p_{v'}$. In other words,

$$\begin{aligned} wp_1 \;=\; & a_1 \prec_{\boldsymbol{i}_1} \cdots \prec_{\boldsymbol{i}_{p_{v'}-1}} \\ & a_{p_{v'}} \prec_{\boldsymbol{i}_{p_{v'}}} \cdots \prec_{\boldsymbol{i}_{p_v-1}} \\ & a_{p_v} \prec_{\boldsymbol{i}_{p_v}} \cdots \prec_{\boldsymbol{i}_{n-1}} a_n, \end{aligned}$$

$$\begin{aligned} wp_2 \;=\; & a_{p_1} \prec_{\boldsymbol{j}_1} \cdots \prec_{\boldsymbol{j}_{v-1}} \\ & a_{p_v} \prec_{\boldsymbol{j}_v} \cdots \prec_{\boldsymbol{j}_{v'-1}} \\ & a_{p_{v'}} \prec_{\boldsymbol{j}_{v'}} \cdots \prec_{\boldsymbol{j}_{m-1}} a_{p_m}. \end{aligned}$$

It is easy to verify that text value τ (defined in Equation 3.2) satisfies wp_1 but it does not satisfies wp_2; a contradiction.

Step 3 (Condition 2) Finally, we will prove that for every $v = 1, \ldots, m-1$, we

have:

$$inf(\boldsymbol{j}_v) \leq inf(\boldsymbol{i}_{p_v}) + \cdots + inf(\boldsymbol{i}_{p_{v+1}}) + p_{v+1} - p_v - 1$$

$$sup(\boldsymbol{j}_v) \text{ is } \begin{cases} \geq \begin{pmatrix} sup(\boldsymbol{i}_{p_v}) + \cdots + \\ sup(\boldsymbol{i}_{p_{v+1}}) + \\ p_{v+1} - p_v - 1 \end{pmatrix} & \begin{array}{l} \text{if all } sup(\boldsymbol{i}_{p_v}), \ldots, \\ sup(\boldsymbol{i}_{p_{v+1}}) \text{ are} \\ \text{different than } \infty \end{array} \\ \infty \quad \text{otherwise} \end{cases}$$

By contradiction, let us assume that there exists a subformula $a_{p_v} \prec_{\boldsymbol{j}_v} a_{p_{v+1}}$ of wp_2 such that

$$inf(\boldsymbol{j}_v) > inf(\boldsymbol{i}_{p_v}) + \cdots + inf(\boldsymbol{i}_{p_{v+1}}) + p_{v+1} - p_v - 1 \tag{3.3}$$

From Step 2, word patterns wp_1 and wp_2 are of the following form:

$$\begin{aligned} wp_1 \;=\; & a_1 \prec_{\boldsymbol{i}_1} \cdots \prec_{\boldsymbol{i}_{p_v-1}} \\ & a_{p_v} \prec_{\boldsymbol{i}_{p_v}} \cdots \prec_{\boldsymbol{i}_{p_{v+1}-1}} \\ & a_{p_{v+1}} \prec_{\boldsymbol{i}_{p_v}} \cdots \prec_{\boldsymbol{i}_{n-1}} a_n, \end{aligned}$$

$$\begin{aligned} wp_2 \;=\; & a_{p_1} \prec_{\boldsymbol{j}_1} \cdots \prec_{\boldsymbol{j}_{v-1}} \\ & a_{p_v} \prec_{\boldsymbol{j}_v} \\ & a_{p_{v+1}} \prec_{\boldsymbol{j}_{v+1}} \cdots \prec_{\boldsymbol{j}_{m-1}} a_{p_m}. \end{aligned}$$

Let us now construct a text value τ' defined as:

$$\begin{aligned}\tau' \;=\;& a_1 \underbrace{\#\cdots\#}_{i_1 \text{ times}} a_2 \cdots \\ & a_{p_v} \underbrace{\#\cdots\#}_{i_{p_v} \text{ times}} a_{p_v+1} \cdots \\ & a_{p_{v+1}-1} \underbrace{\#\cdots\#}_{i_{p_{v+1}-1} \text{ times}} a_{p_{v+1}} \cdots \\ & a_{n-1} \underbrace{\#\cdots\#}_{i_{n-1} \text{ times}} a_n\end{aligned} \tag{3.4}$$

where # is a special word which is not contained in wp_1 and wp_2, and for every s, $1 \leq s \leq n-1$, $i_s = inf(\boldsymbol{i}_s)$ holds. It is easy to verify that τ' satisfies wp_1. Notice that between words a_{p_v} and $a_{p_{v+1}}$ in τ' there are exactly $inf(\boldsymbol{i}_{p_v}) + \cdots + inf(\boldsymbol{i}_{p_{v+1}}) + p_{v+1} - p_v - 1$ words. Therefore, since Expression 3.3 holds, τ' does not satisfy the subformula $a_{p_v} \prec_{\boldsymbol{j}_v} a_{p_{v+1}}$ of wp_2 and thus, it does not satisfy wp_2. Thus, we have $wp_1 \not\models wp_2$ which contradicts our initial assumption.

The proof involving $sup(\boldsymbol{j}_v)$ is similar. It differs only in the way we construct text value τ' (Expression 3.4) and specifically in the values of $i_1, \ldots, i_{n-1}$. We now require that $i_1 \in \boldsymbol{i}_1, \ldots, i_{n-1} \in \boldsymbol{i}_{n-1}$ and for every s, $p_v \leq s \leq p_{v+1}$, we define:

$$i_s = \begin{cases} sup(\boldsymbol{i}_s) & \text{if } sup(\boldsymbol{i}_s) \text{ is different than } \infty \\ sup(\boldsymbol{j}_v) + 1 & \text{otherwise} \end{cases}$$

■

Proposition 3.3.5 *Let wp_1 and wp_2 be proximity word patterns with n and m words respectively. Deciding whether $wp_1 \models wp_2$ can be done in $O(n+m)$ time.*

Let $SAT(\mathcal{WP})$ denote the satisfiability problem for formulas of $\mathcal{WP}$. The following two propositions show that the problems SAT and $SAT(\mathcal{WP})$ are equivalent under polynomial time reductions.

Proposition 3.3.6 *SAT is polynomially reducible to $SAT(\mathcal{WP})$.*

Proof: Trivial by considering propositional variables to be words. ■

Proposition 3.3.7 *$SAT(\mathcal{WP})$ is polynomially reducible to SAT.*

Proof: Let ϕ be a formula of $\mathcal{WP}$. We transform ϕ into an instance ϕ' of SAT as follows. We start with ϕ' being ϕ (words of ϕ play the role of propositional variables in ϕ'). Then, we substitute each proximity word pattern wp of ϕ' by a brand new propositional variable v_{wp}. Finally, we conjoin to ϕ' the following formulas:

- $v_{wp} \implies w$, for each proximity word pattern wp and word w of wp.
- $v_{wp_1} \implies v_{wp_2}$, for each pair of proximity word patterns wp_1, wp_2 such that $wp_1 \models wp_2$.

The above steps can be done in polynomial time because entailment of proximity word patterns can be done in polynomial time (Proposition 3.3.5). It is also easy to see that ϕ is a satisfiable formula of $\mathcal{WP}$ iff ϕ' is a satisfiable formula of Boolean logic. Then, the result holds. ■

Propositions 3.3.6 and 3.3.7 have the following corollary.

Corollary 3.3.1 *Deciding whether a word pattern is satisfiable is a $\mathcal{NP}$-complete problem. Deciding whether a word pattern entails another is a co$\mathcal{NP}$-complete problem.*

Let us close this section by pointing out that satisfiability and entailment of conjunctive word patterns can be done in PTIME.

Proposition 3.3.8 *The satisfiability and entailment problems for conjunctive word patterns can be solved in polynomial time.*

Proof: This is easy to see given Proposition 3.3.5. ■

3.4 Satisfiability and Entailment in $\mathcal{AWP}$

Let $SAT(\mathcal{AWP})$ denote the satisfiability problem for queries of $\mathcal{AWP}$. The following two propositions show that the problems SAT and $SAT(\mathcal{AWP})$ are equivalent under polynomial time reductions.

Proposition 3.4.1 *SAT is polynomially reducible to $SAT(\mathcal{AWP})$.*

Proof: Let ϕ be an instance of SAT (i.e., a Boolean formula). For every propositional variable p in ϕ introduce an attribute A_p. Then, substitute every occurrence of p in ϕ by $A_p =$ "*true*" to arrive at an instance ψ of $SAT(\mathcal{AWP})$. Obviously, ϕ is satisfiable iff ψ is satisfiable. ■

Proposition 3.4.2 *$SAT(\mathcal{AWP})$ is polynomially reducible to SAT.*

Proof: Let ϕ be a query of $\mathcal{AWP}$. Using Proposition 3.1.2, ϕ can easily be transformed into a formula θ which is a Boolean combination of atomic queries. This transformation can be done in time linear in the size of the formula.

The next step is to substitute in θ atomic formulas $A = s$ and $A \sqsupseteq wp$ (where wp is a word or a proximity word pattern) by propositional variables $p_{A=s}$ and $p_{A \sqsupseteq wp}$ respectively to obtain formula θ'. Finally, the following formulas are conjoined to θ' to obtain ψ:

1. If $A = s_1$ and $A = s_2$ are conjuncts of θ' and $s_1 \neq s_2$ then conjoin $p_{A=s_1} \equiv \neg p_{A=s_2}$.
2. If $A = s$ and $A \sqsupseteq wp$ are conjuncts of θ' and $s \models wp$ then conjoin $p_{A=s} \implies p_{A \sqsupseteq wp}$.
3. If $A = s$ and $A \sqsupseteq wp$ are conjuncts of θ' and $s \not\models wp$ then conjoin $p_{A=s} \implies \neg p_{A \sqsupseteq wp}$.
4. If $A \sqsupseteq wp_1$ and $A \sqsupseteq wp_2$ are conjuncts of θ' and $wp_1 \models wp_2$ then conjoin $p_{A \sqsupseteq wp_1} \implies p_{A \sqsupseteq wp_2}$.

The above step can be done in polynomial time because satisfaction and entailment of word patterns in θ can be done in polynomial time. The result for satisfaction is obvious and the result for entailment is from Proposition 3.3.5. It is also easy to see that ϕ is a satisfiable query iff ψ is a satisfiable formula of Boolean logic. Then, the result holds. ■

Propositions 3.4.1 and 3.4.2 have the following corollary.

Corollary 3.4.1 *Deciding whether a query of $\mathcal{AWP}$ is satisfiable is a $\mathcal{NP}$-complete problem. Deciding whether a query of $\mathcal{AWP}$ entails another is a co$\mathcal{NP}$-complete problem.*

The following proposition shows that, as in the case of $\mathcal{WP}$, satisfiability and entailment of conjunctive queries in $\mathcal{AWP}$ can be done in PTIME. This is good news given that conjunctive $\mathcal{AWP}$ queries are typically utilized in implementations such as [106, 123, 202].

Proposition 3.4.3 *The satisfiability and entailment problems for conjunctive $\mathcal{AWP}$ queries can be solved in polynomial time.*

To obtain a more accurate picture of the tractable vs. intractable classes of queries in $\mathcal{AWP}$ one can profitably utilize such results from the propositional satisfiability literature. For example, it is easy to see now that each tractable class C of SAT formulas has a corresponding class C' of tractable formulas of $\mathcal{WP}$ or $\mathcal{AWP}$ if the 2-variable propositional formulas used in the proofs of Propositions 3.3.7 and 3.4.2 belong to C (e.g., this holds for C being the class of propositional formulas with at most two variables using the tractability of 2-SAT).

3.5 Similar Models

In this section we discuss related research which is specific to this chapter and was not covered in Chapter 2. Since formal analysis based on logic and complexity as done in this chapter is not common in Information Retrieval research, this section briefly surveys other data models (and systems) related to the ones studied in this work.

3.5.1 Word Patterns and Proximity Operators

To the best of our knowledge, the papers by Chang and colleagues [49–51], the present chapter and paper [126] are the only comprehensive formal treatments of proximity word patterns in the literature.

Search engines use models similar to $\mathcal{WP}$ and $\mathcal{AWP}$. The most common support for word patterns in search engines includes the ability to combine words using the Boolean operators $\wedge, \vee$ and $\neg$. However, search engines support a version of negation in the form of binary operator $AND\text{-}NOT$ which is essentially set difference, and therefore *safe* in the database-theoretic sense of the term [9]. For example, a search engine query wp_1 $AND\text{-}NOT$ wp_2 will return the set of documents that satisfy wp_1 *minus* these that satisfy wp_2. Note also that the previous work of [49] has *not* considered negation in its word pattern language but has considered negation in the query language which supports attributes (the one that corresponds to our model $\mathcal{AWP}$).

Proximity operators are a useful extension of the concept of "phrase search" used in current search engines. Limited forms of proximity operators have been offered in the past by various search engines of the pre-Google era (e.g., Altavista had an operator $NEAR$ which meant word-distance 10, Lycos had an operator $NEAR$ which meant word-distance 25, and Infoseek used to have a more sophisticated facility). Google supports proximity by the use of operator "$*$" which, when used between two keywords, specifies a minimum distance of one word between them (multiple occurrences of $*$ can also be used to specify a larger minimum distance). The search engine Exalead[3] has an operator NEAR which returns documents that contain given keywords in a vicinity of a fixed number of words, but no ordering of words is supported.

The need to change their index structures and the high computational cost of proximity search, is probably the reason that makes current search engines to limit proximity support to less general operators compared to those used in models $\mathcal{WP}$

[3]Exalead (http://www.exalead.com/) is a search engine developed in France. We mention it here because Exalead is involved in the Quaero project launched in Europe in the summer of 2005 as the European response to Google.

and $\mathcal{AWP}$.

Proximity operators have also been implemented in other systems such as free-WAIS [159] and INQUERY [42]. There are also advanced IR models such as the model of proximal nodes [148] with proximity operators between arbitrary structural components of a document (e.g., paragraphs or sections). Data models and query languages for full-text extensions to XML e.g., TeXQuery [20] is the most recent area of research where proximity operators have been used.

Proximity word patterns can also be viewed as a particular kind of *order constraints* in the sense of constraint networks [68] and databases [169]. There are many papers that discuss algorithms and complexity of various kinds of order constraints e.g., gap-order constraints [170] or temporal constraints [69, 122]. The algorithms and complexity results regarding $\mathcal{WP}$ can also be viewed as a contribution to this research area.

3.5.2 Other Operators from Information Retrieval

The data model $\mathcal{AWP}$ discussed in Section 3.1 complements recent proposals for representing and querying textual information in publish/subscribe systems [43, 44] by using linguistically motivated concepts such as *word* and traditional IR operators (instead of strings and operators such as string containment [43, 44]). The methodology and techniques of this work can be used to study the complexity of satisfiability and entailment for the subscription query language of [43] and we expect the complexity results to be similar.

In Section 3.2 we have extended the model $\mathcal{AWP}$ by introducing a "similarity" operator based on the IR vector space model [24]. The similarity concept of the new model $\mathcal{AWPS}$ has in the past been used in database systems with IR influences (e.g., WHIRL [58]) and more recently in XML-based query languages e.g., ELIXIR [53], XIRQL [85] and XXL [192].

3.6 Conclusions

In this chapter we presented the models $\mathcal{WP}$, $\mathcal{AWP}$ and $\mathcal{AWPS}$ that will be used later in Chapters 4, 5 and 6. We studied the model theory of $\mathcal{WP}$ and $\mathcal{AWP}$ focusing on questions related to satisfiability and entailment. We showed that the satisfiability problem for queries in $\mathcal{WP}$ and $\mathcal{AWP}$ is $\mathcal{NP}$-complete and the entailment problem is co$\mathcal{NP}$-complete. We also discussed cases where these problems can be solved in polynomial time.

In the next chapter, we present a P2P architecture to support retrieval and filtering functionality, discuss the basics of the API to support this architecture and present the protocols that orchestrate the peers.

Chapter 4

An Architecture for Peer-to-Peer Web Search

In this chapter we present an architecture that is able to support full-fledged Web information retrieval and filtering in a single unifying framework. This architecture is based on ideas from traditional distributed IR and recent work on P2P networks. Our architecture, called LibraRing (from the words *libra*ry and *ring*), is hierarchical like the ones in [115, 130] but uses a DHT to achieve robustness, fault-tolerance and scalability in its routing and meta-data management layer. DHTs are the second generation *structured* P2P overlay networks devised as a remedy for the known limitations of earlier P2P networks such as Napster and Gnutella (see Section 2.1 for literature related to P2P systems and DHTs).

As we have already said in Section 1.1, there are two kinds of basic functionality that we expect this architecture to offer: information retrieval and publish/subscribe. In an IR scenario, a user can pose a *query* (e.g., "I am interested in papers on bio-informatics") and the system returns information about matching resources. In a pub/sub scenario, a user posts a *subscription* to the system to receive notifications whenever certain events of interest take place (e.g., when a paper on bio-informatics becomes available).

The main components of our architecture are *super-peers, clients* and *providers.* Providers are used to expose the content of information sources to the network, while

clients are used by information consumers. Super-peers form an overlay network that offers a robust, fault-tolerant and scalable means for routing messages and managing resource meta-data and queries. The main architectural contribution of our work is the extension of the DHT Chord protocols [180] with IR and pub/sub functionality in the context of a super-peer network. To validate our approach, we apply it in a DL environment and discuss the protocols for offering retrieval and filtering functionality in a two-tier architecture. Although the protocols we present here are tailored for a super-peer architecture, our ideas can be easily applied in a pure P2P environment, as we will show in Chapter 5. The results of this chapter have been published in [202].

The data models and query language we will choose will have a serious effect on the DHT protocols because the DHT is the layer in which publications, queries and subscriptions live (are indexed). In the rest of this chapter, we assume that publications and subscriptions will be expressed using model $\mathcal{AWPS}$, presented in the previous chapter.

The research presented in this chapter is a continuation of our previous work on the DL information alert architecture DIAS [123] and the system P2P-DIET [104–106]. The main difference of the current work from [105, 123] is the definition of an architecture for content-based web search and brand new protocols that are extensions of DHTs and are able to support retrieval and filtering tasks in a distributed environment.

The organisation of this chapter is as follows. In Section 4.1 we discuss the proposed architecture and an application scenario based on digital libraries. Subsequently, Section 4.2 discusses some useful extensions to the Chord API, while Section 4.3 presents the LibraRing protocols. Finally, Section 4.4 summarises our approach and concludes the chapter.

4.1 The LibraRing Architecture

A high-level view of the LibraRing architecture is shown in Figure 4.1. Nodes can implement any of the following types of services: *super-peer service, provider*

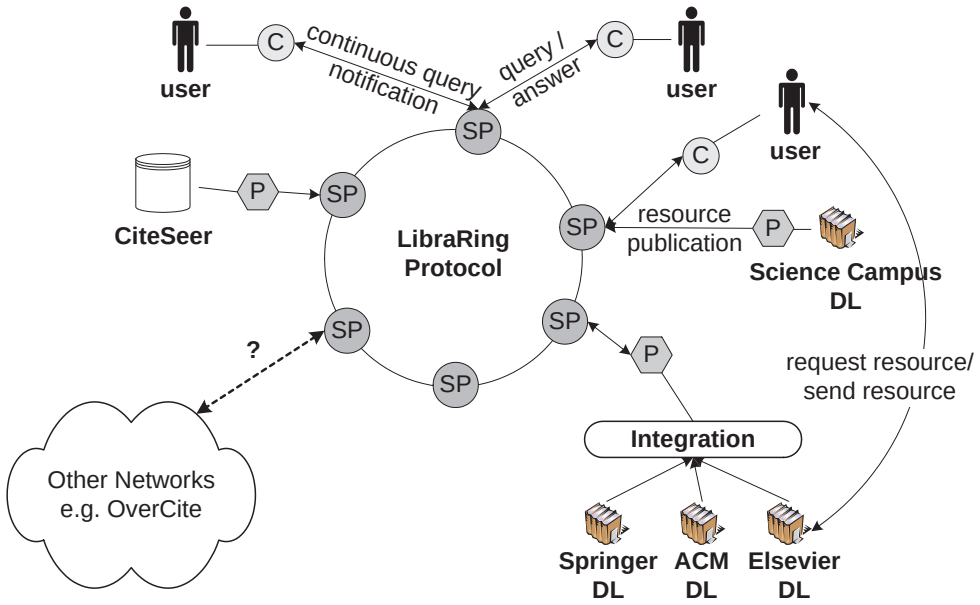


Figure 4.1: The architecture of LibraRing

service and *client service*.

Super-peer service. Nodes implementing the super-peer service (*super-peers*) form the message routing layer of the network. Each super-peer is responsible for serving a fraction of the clients by storing continuous queries and resource publications, answering one-time queries, and creating notifications. The super-peers run a DHT protocol which is an extension of Chord. The role of the DHT in LibraRing is very important. First of all, it acts as a rendezvous point for information producers (providers) and information consumers (clients). Secondly, it serves as a robust, fault-tolerant and scalable routing infrastructure. When the number of super-peers is small, each node can easily locate others in a single hop by maintaining a full routing table. When the super-peer network grows in size, the DHT provides a scalable means of locating other nodes. Finally, by serving as a global metadata index that is partitioned among super-peers, the DHT facilitates the building of a distributed metadata repository that can be queried efficiently.

Client service. Nodes implementing the client service are called *clients*. A client connects to the network through a single super-peer node, which is its *access point*. Clients can connect, disconnect or even leave the system silently at any time. Clients are information consumers: they can pose one-time queries and receive

answers, subscribe to resource publications and receive notifications about published resources that match their interests. If clients are not on-line, notifications matching their interests are stored by their access points and delivered once clients reconnect. Resource requests are handled directly by the client that is the owner of the resource.

Provider service. This service is implemented by information sources that want to expose their contents to the clients of LibraRing. A node implementing the provider service (*provider*) connects to the network through a super-peer which is its access point. To be able to implement this service, an information source should create meta-data for the resources it stores using data model $\mathcal{AWPS}$, and publish it to the rest of the network using its access point.

As an example of an application scenario, let us consider a university with 3 geographically distributed campuses (Arts, Sciences and Medicine) and a local digital library in each campus (see Figure 4.1). Each campus maintains its own super-peer, which provides an access point for the provider representing the campus digital library, and the clients deployed by users. A university might be interested in making available to its students and staff, in a timely way, the content provided by other publishers (e.g., CiteSeer, ACM, Springer, Elsevier). Figure 4.1 shows how our architecture can be used to fulfill this requirement. An integration layer is used to unify different types of DLs. At this level, we also expect to see *observer modules* (as in [76]) for information sources that do not provide their own alerting service. This modules will query the sources for new material in a scheduled manner and inform providers accordingly.

Wrappers of heterogeneous sources of information are also able to expose the content of the wrapped sources by implementing a provider service. As also shown in Figure 4.1, integration software build to provide a common view of heterogeneous content sources will be able to publish the content of its sources to the rest of LibraRing network. Additionally, content repositories such as CiteSeer or Google Scholar can use one or multiple provider services to publish their content to the rest of the network. Interoperability with other networks (e.g. OverCite [183]) serving a similar purpose is a desirable feature in the design of any future digital library. In the context of this work however we do not explore this dimension in any way. Some

nice ideas in the field of interoperability of DHTs have been recently presented in [6].

The above distributed resource sharing scenario appears in many application areas beyond digital libraries (e.g., financial trading or e-commerce where incoming information and queries/profiles might refer to prices, news dissemination where incoming information and queries/profiles are textual, service notification where incoming information and queries/profiles are defined in an appropriate service description language). In this work we are especially concerned with *textual information*, and concentrate on the protocols that regulate peer interactions in the architecture described earlier.

4.2 Extensions to the Chord API

To facilitate message sending between nodes we will use the function `send(`msg, I`)` to send message msg from some node to node $successor(I)$, where I is a node identifier. Function `send()` is similar to Chord function `lookup(`I`)` [180], and costs $O(logN)$ overlay hops for a network of N nodes. When function `send(`msg, I`)` is invoked by node S, it works as follows. S contacts S', where $id(S')$ is the greatest identifier contained in the finger table of S, for which $id(S') \leq I$ holds. Upon reception of a `send()` message by a node S, I is compared with $id(S)$. If $id(S) < I$, then S just forwards the message by calling `send(`msg, I`)` itself. If $id(S) \geq I$, then S processes msg since it is the *intended recipient*.

Our protocols described in Section 4.3 also require that a node is capable of sending the *same* message to a group of nodes. This group is created dynamically each time a resource publication or a query submission takes place, so multicast techniques for DHTs such as [16] are not applicable. The obvious way to handle this over Chord is to create k different `send()` messages, where k is the number of different nodes to be contacted, and then locate the recipients of the message in an *iterative* fashion using $O(k \log N)$ messages.

We have also designed and implemented function `multiSend(`msg, L`)`, where L is a list of k identifiers, that can be used to send message msg to the k elements of L in a

recursive way. When function `multiSend()` is invoked by node S, it works as follows. Initially S sorts the identifiers in L in ascending order clockwise starting from $id(S)$. Subsequently S contacts S', where $id(S')$ is the greatest identifier contained in the finger table of S, for which $id(S') \leq head(L)$ holds, where $head(L)$ is the first element of L. Upon reception of a `multiSend()` message, by a node S, $head(L)$ is compared with $id(S)$. If $id(S) < head(L)$, then S just forwards msg by calling `multiSend()` again. If $id(S) \geq head(L)$, then S processes msg since this means that it is *one* of the intended recipients contained in list L (in other words, S is responsible for key $head(L)$). Then S creates a new list L' from L in the following way. S deletes all elements of L that are smaller or equal to $id(S)$, starting from $head(L)$, since S is responsible for them. In the new list L' that results from these deletions, we have that $id(S) < head(L')$. Finally, S forwards msg to node with identifier $head(L')$ by calling `multiSend(`msg, L'`)`. This procedure continues until all identifiers are deleted form L.

4.3 The LibraRing Protocols

In this section we describe in detail the way clients, providers and super-peers join and leave the network. We also describe resource publication and query submission protocols. We use functions $key(n)$, $ip(n)$ and $id(n)$ to denote the key, the IP address and the identifier of node n respectively. Keys for nodes are generated by hashing a concatenation of their IP address, port number and the timestamp of their first connection. Node identifiers are used by super-peers to define their position in the DHT, and are generated using the cryptographic hash function $H()$ utilised by the routing infrastructure and their IP address and port. Keys are used for uniquely identifying nodes, while identifiers are used for participating in the structure overlay.

4.3.1 Client Join

The *first time* that a client C wants to connect to the LibraRing network, it has to follow the join protocol. C must find the IP address of a super-peer S using out-of-band means (e.g., via a secure web site that contains IPs for the super-peers that are

currently online). C sends to S message NewClient($key(C), ip(C)$) and S adds C in its *clients table* (CT), which is a hash table used for identifying the peers that use S as their *access point*. $key(C)$ is used to index clients in CT, while each CT slot stores contact information about the client, its status (connected/disconnected) and its stored notifications[1]. Additionally, S sends to C an acknowledgement message AckNewClient($id(S)$). Once C has joined, it can use the connect/disconnect protocol (to be described below) to connect and disconnect from the network.

Providers use a similar protocol to join a LibraRing network.

4.3.2 Client Connect/Disconnect

When a client C wants to connect to the network, it sends a message Connect-Client($key(C), ip(C), id(S)$) to its access point S. If $key(C)$ exists in the CT of S, C is marked as connected and stored notifications are forwarded to it. If $key(C)$ does not exist in CT, this means that S was not the access point of C the last time that C connected (Section 4.3.7 discusses this case).

When a client C wants to disconnect, it sends a message DisconnectClient($key(C), ip(C)$) to its access point S. S marks C as disconnected in its CT without removing information related to C, since this information will be used to create stored notifications for C while C is not online (see Section 4.3.6).

Providers connect to and disconnect from the network in a similar way.

4.3.3 Resource Indexing

When a provider P decides to index a resource, metadata for this resource are created, and sent in the network where they will be indexed at the responsible nodes, while the resource itself remains at the provider. This section describes the indexing of the metadata of a resource in the overlay.

A resource is indexed in three steps. In the first step a provider P constructs a

[1]Notifications created for C while it was not online, and are stored in order to be delivered upon connection. These ideas originally appeared in system P2P-DIET [104–106] and are also briefly discussed in Section 4.3.6.

publication $p = \{(A_1, s_1), (A_2, s_2), \ldots, (A_n, s_n)\}$ (the resource description) and sends message PUBRESOURCE$(key(P), ip(P), key(p), p)$ to its access point S.

In step two, S forwards p to the appropriate super-peers as follows. Let $D_1, \ldots, D_n$ be the sets of *distinct* words in $s_1, \ldots, s_n$. Then p is sent to *all* nodes with identifiers in the list $L = \{H(w_j) : \ w_j \in D_1 \cup \cdots \cup D_n\}$. The protocol guarantees that L is a superset of the set of identifiers responsible for queries that match p. Subsequently, S removes duplicates and sorts L in ascending order clockwise starting from $id(S)$. In this way we obtain less identifiers than the distinct words in $D_1 \cup \cdots \cup D_n$, since a super-peer may be responsible for more than one words contained in the document. Having obtained L, S indexes p by creating message $msg =$INDEXRESOURCE$(ip(P), key(P), ip(S), key(p), p)$, and calling function `multiSend(`msg, L`)`.

Finally, in the third step, each super-peer S' that receives this message stores p in an *inverted index* that will facilitate matching against one-time queries that will arrive later on at S'.

To facilitate faster publication removal and update S stores the IP addresses and identifiers of all the super-peers indexing publications that belong to its providers. This information is found from the acknowledgement messages sent by the super-peers that index each publication.

4.3.4 Submitting an One-Time Query

In this section we show how to answer one-time queries containing Boolean and vector space atomic queries or only vector space atomic queries. The first type of queries is always indexed under its Boolean part. Let us assume that a client C wants to submit a query q of the form $\bigwedge_{i=1}^{m} A_i = s_i \wedge \bigwedge_{i=m+1}^{n} A_i \sqsupseteq wp_i \wedge \bigwedge_{i=n+1}^{k} A_i \sim_{a_i} s_i$. The following three steps take place. In step one, C sends to its access point S message SUBMITQ$(key(C), ip(C), key(q), q)$.

In the second step, S randomly selects a single word w contained in any of the text values $s_1, \ldots, s_m$ or word patterns $wp_{m+1}, \ldots, wp_n$ and computes $H(w)$ to obtain the identifier of the super-peer storing publications that can match q. Then, it

sends message msg =PoseQuery$(ip(C), key(C), ip(S), key(q), q)$ by calling function `send(`$msg, H(w)$`)`.

If q is of the form $A_{n+1} \sim_{a_1} s_1 \wedge ... \wedge A_n \sim_{a_n} s_n$ then step two is modified as follows. Let $D_1, \ldots, D_n$ be the sets of *distinct* words in $s_1, \ldots, s_n$. q has to be sent to *all* super-peers with identifiers in the list $L = \{H(w_j) : w_j \in D_1 \cup \cdots \cup D_n$. To do so, S removes duplicates, sorts L in ascending order clockwise starting from $id(S)$ and sends message msg =PoseQuery$(ip(C), key(C), ip(S), key(q), q)$ by calling `multiSend(msg,L)`.

In step three, each super-peer that receives a one-time query q, matches it against its local publication store to find out which providers have published documents that match q and delivers answers as discussed in Section 4.3.6.

4.3.5 Publish/Subscribe Functionality

This section describes how to extend the protocols of Sections 4.3.4 and 4.3.3 to provide pub/sub functionality. To index a continuous query cq the one-time query submission protocol needs to be *modified*. The first two steps are identical, while the third step is as follows. Each super-peer that receives cq, stores cq in its local continuous query data structures to match it against incoming publications. A super-peer S uses a hash table to index all the atomic queries of cq, using as key the attributes $A_1, \ldots, A_k$. To index each atomic query, three different data structures are also used: (i) a hash table for text values $s_1, \ldots, s_m$, (ii) a trie-like structure that exploits common words in word patterns $wp_{m+1}, \ldots, wp_n$, and (iii) an inverted index for the most "significant" words in text values $s_{n+1}, \ldots, s_k$. S' utilises these data structures at filtering time to find quickly all continuous queries cq that match an incoming publication p. This is done using an algorithm that combines BestFitTrie (described in detail in Chapter 6 and also in [200]) and SQI [211].

To index a resource, the protocol of Section 4.3.3 needs to be *extended*. The first two steps are identical, while in the third step, each super-peer that receives p matches it against its local continuous query database using the algorithms BestFitTrie and SQI.

To facilitate faster continuous query removal and update S stores the IP addresses and identifiers of all the super-peers indexing continuous queries that belong to its clients. This information is found from the acknowledgement messages sent by the super-peers that index each continuous query.

4.3.6 Notification Delivery

Assume a super-peer S that has to deliver a notification n for a continuous query cq to client C. S creates message msg =NOTIFICATION$(ip(P), key(P), pid(p), qid(cq))$, where P is the provider that published the matching resource and sends it to C. If C is not online, then S sends msg to S', where S' is the access point of C, using $ip(S')$ associated with cq. S' stores msg, to deliver it to C upon reconnection. If S' is also off-line msg is sent to the $successor(id(S'))$, by calling function `send(`$msg, successor(id(S'))$`)`. Answers to one-time queries are handled in a similar way. In case that more that one answers or notifications have to be delivered, function `multiSend()` is used.

4.3.7 Super-Peer Join/Leave

To join the LibraRing network, a super-peer S must find the IP address of another super-peer S' using out-of-band means. S creates message NEWSPEER$(id(S), ip(S))$ and sends it to S' which performs a lookup operation by calling `lookup(id(S))` to find $S_{succ} = successor(id(S))$. S' sends message ACKNEWSPEER$(id(S_{succ}), ip(S_{succ}))$ to S and S updates its successor to S_{succ}. S also contacts S_{succ} asking its predecessor and the data that should now be stored at S. S_{succ} updates its predecessor to S, and answers back with the contact information of its previous predecessor, S_{pred}, and all continuous queries and publications that were indexed under key k, with $id(S) \leq k < id(S_{pred})$. S makes S_{pred} its predecessor and populates its index structures with the new data that arrived. After that S populates its finger table entries by repeatedly performing lookup operations on the desired keys.

When a super-peer S wants to leave LibraRing network, it constructs message DISCONNECTSPEER$(id(S), ip(S), id(S_{pred}), ip(S_{pred}), data)$, where $data$ are all the

continuous queries, published resources and stored notifications of off-line peers that S was responsible for. Subsequently, S sends the message to its successor S_{succ} and notifies S_{pred} that its successor is now S_{succ}. Clients that used S as their access point connect to the network through another super-peer S'. Stored notifications can be retrieved through $successor(id(S))$.

4.4 Conclusions

We have presented an architecture especially designed for textual information retrieval and filtering that uses a variation of the Chord DHT as its routing infrastructure. We have defined an API that is used in the context of this architecture and enables the efficient dissemination of the available resources to the participating peers. Finally, we have presented a suite of protocols that define the behaviour of the three types of peers that populate the system.

In the next chapter we put our focus on the filtering problem, and design efficient and scalable algorithms that support pub/sub functionality in a distributed, dynamic environment. The reader interested for work on the retrieval case is referred to the Section 2.3, where different architectures and models for P2P IR are surveyed.

Chapter 5

Protocols for Distributed Information Filtering

In the previous chapter, we presented an architecture that unifies information retrieval and filtering using a DHT as the routing substrate, and discussed the protocols that regulate peer interactions and provide this kind of functionality. In this chapter, we focus on the case of information filtering and present an in-depth investigation of the pub/sub protocols, collectively called *DHTrie*, that extend the Chord protocols with pub/sub functionality assuming that publications and subscriptions are expressed in the model $\mathcal{AWPS}$. The results of this chapter have been published in [196, 197, 199, 201, 203].

In a pub/sub environment publications typically involve contacting a large set of nodes. To achieve this in an efficient and scalable way, we have designed and implemented three methods that target low network traffic and low latency. In combination with these methods, we introduce a simple routing table that uses only local information and manages to reduce network traffic to a factor of 9. We justify our solution by evaluating the DHTrie protocols experimentally in a distributed digital library scenario with hundreds of thousands of nodes and millions of user profiles. Our experiments show that the DHTrie protocols are *scalable*: the number of messages it takes to publish a document and notify interested subscribers remains almost constant as the network grows, while publication latency is kept low. More-

over, the increase in message traffic shows little sensitivity to increase in document size.

Since probability distributions associated with publication and query elements are expected to be skewed in typical pub/sub scenarios, achieving a *balanced load* is an important problem. We study three important cases of load balancing for DHTrie, namely *query*, *routing* and *filtering* load balancing, and present a new algorithm which is also applicable to the standard DHT look-up problem.

This chapter consists of four sections. Section 5.1 presents the pub/sub protocols and discusses the use of an extra routing table, named FCache, that reduces network traffic using only local information available at each node. Section 5.2 presents an extensive experimental evaluation of the protocols, while Section 5.3 studies the problem of load balancing and proposes a new algorithm that is evaluated experimentally. Finally, Section 5.4 summarises our results and concludes the chapter.

5.1 The DHTrie Protocols

We implement pub/sub functionality by a set of protocols called the *DHTrie protocols* (from the words DHT and trie). The DHTrie protocols use *two levels of indexing* to store queries submitted by clients. The first level corresponds to the partitioning of the global query index to different nodes using DHTs as the underlying infrastructure. Each node is responsible for a fraction of the submitted user queries through a mapping of attribute values to node identifiers. The DHT infrastructure is used to define the mapping scheme and also manages the routing of messages between different nodes. The set of protocols that regulate node interactions are described in the next sections.

The second level of our indexing mechanism is managed locally by each node and is used for indexing the user queries the node is responsible for. In this level, each node uses a hash table to index all the atomic queries contained in a complex query by using their attribute name as the key. For each atomic Boolean query, the hash table points to a *trie*-like structure that exploits *common words* and a hash table that indexes text values in equalities. Additionally, for atomic VSM queries

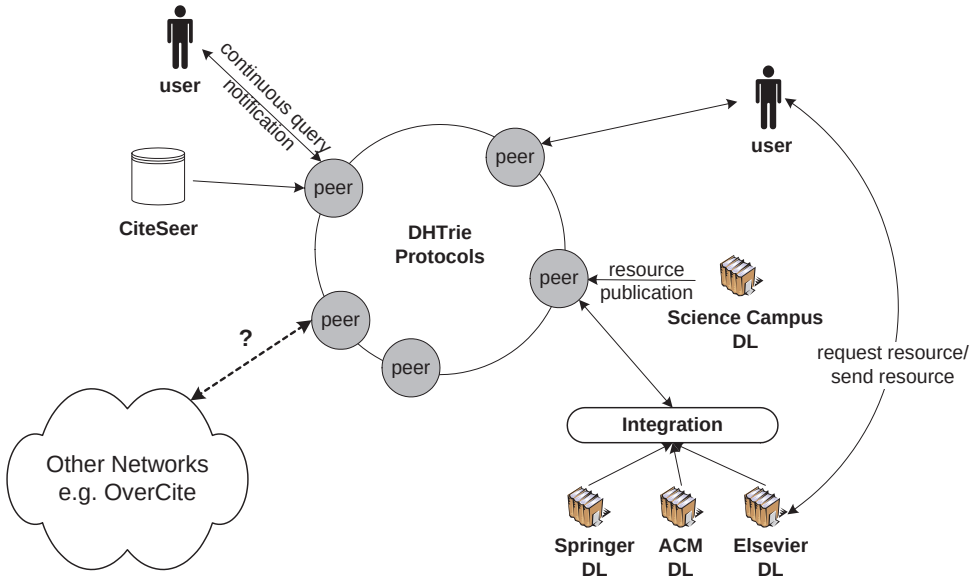


Figure 5.1: Pub/sub functionality for a pure P2P architecture over a structured overlay

an inverted index for the most "significant" query words is used as in [211]. The details of local indexing are presented in the next chapter and also in [200].

VSM relies on term frequencies (tf) and inverse document frequencies (idf) to compute the vector representation of a text value. The computation of idf in an IR or pub/sub scenario needs global statistical information. [64] has shown that in IR scenarios it is enough to have an approximation of the exact idf values. In [188] each peer uses a set of randomly chosen peers to collect such statistics and merge the results to create an approximation of the global idf values. These statistics are updated periodically using sampling. Recent approaches have revisited the problem of distributed statistics maintenance. Minerva [34] assigns each vocabulary term to a node using consistent hashing and devises a heartbeat algorithm that updates word statistics collected at each node. Furthermore, [28] deals with the problem of bulk documents insertions and statistics maintenance in digital library environments and [134] uses highly discriminative keys to reduce postings lists in peers. It is still an open problem how to achieve this in a pub/sub scenario, although the ideas of [28, 34, 64, 188] are relevant in this context as well. We are currently working on this problem and expect to report our results in the future.

In this chapter we will focus on the first level of indexing and the protocols that regulate node interactions. Nodes in our scenario can be super-peers that

participate in the hierarchical architecture of Figure 4.1 discussed in the previous chapter, or peers in a structured overlay without hierarchical distinctions between them as in the architecture shown in Figure 5.1. Thus the protocols presented here are extensions and improvements of the general protocols for providing the filtering functionality presented in Section 4.3 and can be applied on any type of overlay network (hierarchical or not) implemented over Chord. Finally we have to note that the local indexing algorithms used in each node and their experimental evaluation are thoroughly discussed in the next chapter and also in [200, 211].

5.1.1 The Subscription Protocol

Let us assume that a node P wants to submit a query q that contains atomic queries with equality, containment (i.e., Boolean atomic queries) and similarity (i.e., VSM atomic queries) operators of the form:

$$A_1 = s_1 \ \wedge \ ... \ \wedge \ A_m = s_m \ \wedge$$
$$A_{m+1} \sqsupseteq wp_{m+1} \ \wedge \ ... \ \wedge A_n \sqsupseteq wp_n \ \wedge$$
$$A_{n+1} \sim_{a_{n+1}} s_{n+1} \ \wedge \ ... \ \wedge \ A_k \sim_{a_k} s_k$$

To do so, P randomly selects a single word w contained in any of the text values $s_1, \ldots, s_m$ or word patterns $wp_{m+1}, \ldots, wp_n$ and computes $H(w)$ to obtain the identifier of the node that will be responsible for query q. Then P creates message FWDQUERY$(id(P), ip(P), qid(q), q)$, where $qid(q)$ is a unique query identifier assigned to q by P and $ip(P)$ is the IP address of P. This message is then forwarded in $O(logN)$ steps to the node with identifier $H(w)$ using the routing infrastructure of the DHT. This forwarding is done using the DHT lookup function to locate $successor(H(w))$, which is then directly contacted by P. In this way, queries of this type are always indexed under their Boolean part to save message traffic, since they need to be stored at a single node. Notice also that both $id(P)$ and $ip(P)$ need to be sent to the node that will store the query to facilitate notification delivery.

When P wants to submit a query q of the form $A_{n+1} \sim_{a_1} s_1 \ \wedge \ ... \ \wedge \ A_n \sim_{a_n} s_n$ (i.e., with a VSM part only), it sends q to *all* nodes in the list $L = \{H(w_j) : w_j \in D_1 \cup \cdots \cup D_n\}$, where $D_1, \ldots, D_n$ are the sets of *distinct* words in text values

$s_1, \ldots, s_n$. In contrast to queries with a Boolean part, queries with a VSM part *only* need to be stored in all the nodes computed as above in order to ensure correctness of the filtering process. Sending the same message to more than one recipients is discussed in detail in the next section, where publication forwarding poses the same problem.

When a node P' receives a message FwdQuery containing q, it stores q using the second level of our indexing mechanism. P' uses a hash table to index all the atomic queries of q, using as key the attributes $A_1, \ldots, A_k$. To index each atomic query, three different data structures are also used: (i) a hash table for text values $s_1, \ldots, s_m$, (ii) a trie-like structure that exploits common words in word patterns $wp_{m+1}, \ldots, wp_n$, and (iii) an inverted index for the most "significant" words in text values $s_{n+1}, \ldots, s_k$. P' utilises these data structures at filtering time to find quickly all queries q that match an incoming publication p. This is done using an algorithm that combines algorithms BestFitTrie [200] and SQI [211]. The details of local storage and indexing are discussed thoroughly in Chapter 6 and [200].

5.1.2 The Publication Protocol

Resource publication protocols presented here are extensions and modifications of the general protocols presented in Section 4.3.5. Here we revisit the protocols without the need to support a hierarchical architecture and propose optimisations to network traffic and publication latency.

Publication of a resource involves sending the same message to a group of nodes that is not known a priori. To tackle this problem we have designed and implemented three methods: (i) the iterative method, which is the standard way to contact a number of different nodes over Chord, (ii) the recursive method, which creates a single message with all the recipients contained in a sorted list and works its way around the identifier space until all recipients have been contacted, and (iii) the hybrid method which uses machinery from the two previous methods to optimise network traffic and latency. Work presented on a technical report [102] has considered variations of the first two methods and tried to present the tradeoff between implementing multicast

functionality at different levels of the DHT architecture. This approach however tackled content-based multicast from a physical network viewpoint, and focused on network-centric metrics. Thus the results of the report are not directly applicable to our scenario, where overlay hops and publication latency is in question.

The publication protocol essentially involves sending the same message to a group of other nodes, namely those that are responsible for the distinct words contained in the text values of the different attributes of p. In this way, when a node P wants to publish a resource, it first constructs a publication $p = \{(A_1, s_1), (A_2, s_2), \ldots, (A_n, s_n)\}$ (the resource description). Let $D_1, \ldots, D_n$ be the sets of *distinct* words in $s_1, \ldots, s_n$. Then, publication p has to be propagated to *all* nodes responsible for identifiers in the list

$$L = \{H(w_j) : \ w_j \in D_1 \cup \cdots \cup D_n\}. \tag{5.1}$$

The subscription protocol guarantees that L is a superset of the set of identifiers responsible for queries that match p. To propagate publication p in the DHT, P removes duplicates from L and sorts it in ascending order clockwise starting from $id(P)$. In this way we obtain at most as many identifiers as the distinct words in $D_1 \cup \cdots \cup D_n$, since a node may be responsible for more than one of the words contained in the document. Contacting all the nodes responsible for the keys in list L is method specific and is described below in detail.

The Iterative Method

Each node P that uses the iterative method to contact the recipients in list L, constructs a message FWDRESOURCE$(id(P), pid(p), p, id(P'))$ for each identifier $id(P')$ contained in L, where $pid(p)$ is a unique metadata identifier assigned to p by P. Then it utilises the *lookup*() procedure provided by Chord to locate node P' and sends it the FWDRESOURCE$(id(P), pid(p), p, id(P'))$ message. This is repeated for all the identifiers in L in an *iterative* way. Using this method, P needs $O(h \log N)$ messages, where h is the number of different nodes to be contacted. Figure 5.2 illustrates graphically the publication of a resource to three recipients using the iterative

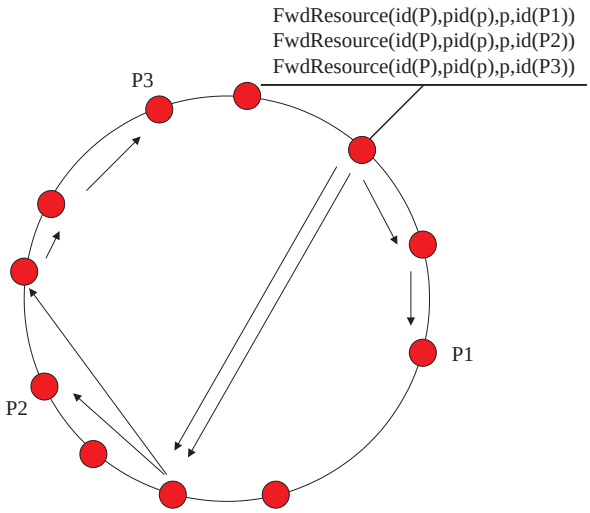


Figure 5.2: An example of the iterative method

method.

The Recursive Method

Using the iterative method has an obvious disadvantage; the same routing request is asked over and over of the same node, causing high network traffic. This is the reason for designing the recursive method. The idea behind the recursive method is to pack messages together and exploit other nodes' routing tables to reduce network traffic. Publishing a resource using the recursive method is as follows.

Having obtained L, P creates a message FwdResource$(id(P), pid(p), p, L)$, where $pid(p)$ is a unique metadata identifier assigned to p by P, and sends it to node with identifier equal to $head(L)$ (the first element of L). This forwarding is done by the following *recursive* way: message FwdResource is sent to a node P', where $id(P')$ is the greatest identifier contained in the finger table of P, for which $id(P') \leq head(L)$ holds.

Upon reception of a message FwdResource by a node P, $head(L)$ is checked. If $id(P) < head(L)$ then P just forwards the message as described in the previous paragraph. If $id(P) \geq head(L)$ then P makes a copy of the message, since this means that P is one of the intended recipients contained in list L (in other words P is responsible for key $head(L)$). Subsequently, the publication part of this message

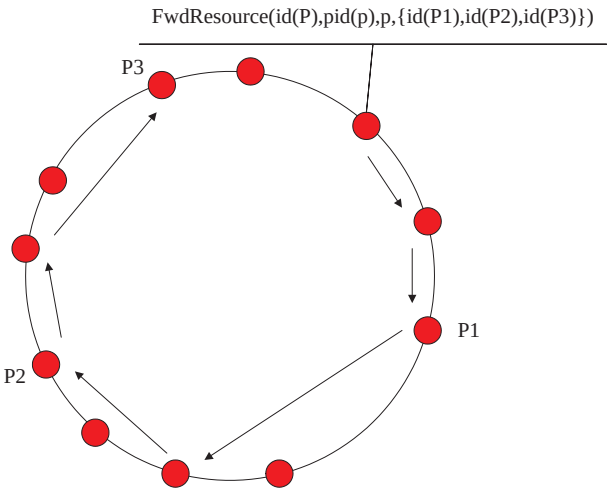


Figure 5.3: An example of the recursive method

is matched with the node's local query database using the algorithms described in detail in Chapter 6 and the appropriate subscribers are notified. Additionally list L is modified to L' in the following way. P deletes all elements of L that are smaller than $id(P)$ starting from $head(L)$, since all these elements have P as their intended recipient. In the new list L' that results from these deletions we have that $id(P) < head(L')$. This happens because in the general case L may contain more than one node identifiers that are managed by P (these identifiers are all located in ascending order at the beginning of L). Finally, P forwards the message to node with identifier $head(L')$. Figure 5.3 presents the publication of a resource to three recipients using the recursive method.

The Hybrid Method

The idea behind the recursive method is to pack messages together and exploit other nodes' routing tables to reduce network traffic. This however comes at the cost of high latency. If the recipients list is long, then the last recipient has to wait for a long time until he is notified about the publication, which in turn causes delays in the notification of the interested subscribers. The iterative method on the other hand tries to optimise latency since no recipients lists are used and the delay to deliver a message is logarithmic in the size of the network. This of course comes at the price of high network traffic as it will be shown in Section 5.2.

To tackle this tradeoff, we designed and implemented a *hybrid* approach that tries to combine the benefits of the two previous methods. The idea behind the hybrid method is to design a tunable alternative that will provide fast delivery of messages at low network cost. To achieve this, we use smaller recipients lists and messages sent in an iterative way. The hybrid method works as follows.

Having obtained L, node P then uses it to create recipients lists, of size at least $S-\delta$ and at most $S+\delta$, where S is a tunable parameter called *desired recipients list size* and δ is the *tolerance factor*, used to force the creation of lists with size close to S (because of the nature of the algorithm creating lists with size exactly S is not always possible). In this way the number of lists created is at least $h_{min} = |L|/(S+\delta)$ and at most $h_{max} = |L|/(S-\delta)$. Each entry $id(P')$ in the finger table of P is used to create one or more lists in the following way. Consider two consecutive entries in the finger table of P, say $id(P')$ and $id(P'')$. Starting from $id(P')$, P scans L and collects all the recipients with identifier greater than $id(P')$ and smaller than $id(P'')$ to create list L_1. If $|L_1| < S-\delta$, P continues to scan L for recipients with identifier greater than $id(P'')$ until $S-\delta \leq |L_1| \leq S+\delta$. On the other hand if $|L_1| > S+\delta$ then list L_1 is divided to lists with size S. This process continues for all the entries in the finger table of P or until list L is empty. Typically finger entries with higher index number have more than one lists associated with them (remember that entries in the finger table of a node points to exponentially increasing distances away from the node, which means that typically the distance between the $i-1$-th and i-th entry is shorter than the distance between the i-th and $i+1$-th entry).

For each one of the lists $L_1 \ldots L_h$ created by the above procedure, a message FwdResource$(id(P), pid(p), p, L_i)$, with $1 \leq i \leq h$, is constructed and is iteratively sent to $head(L_i)$. Since each message contains a list of recipients, the recursive method is used to forward the message to the rest of the nodes in list L_i. This usage of short recipients lists together with the iterative way of sending these lists justifies the hybrid nature of the algorithm. As we will show in Section 5.2, this method manages to minimise latency while keeping message traffic low. This is achieved by creating small recipients lists that exploit routing information located in the finger table of the message initiator. A simpler approach to the hybrid algorithm would be

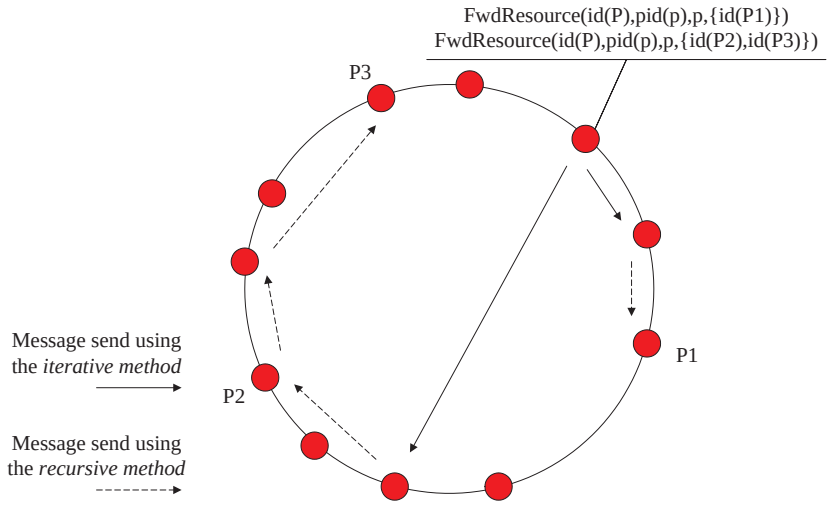


Figure 5.4: An example of the hybrid method

to split the initial list L to a number of smaller fixed size lists according to S, but this would not allow us to use the routing information from the finger table of the message initiator. Figure 5.4 illustrates graphically the publication of a resource to three recipients using the hybrid approach.

Finally, notice that the publication (and also the subscription) protocol indexes equalities using a single word contained in the text value, contrary to the standard way that would index the entire text value in the DHT. This is done to avoid sending extra network messages for each publication to discover matching equalities. False positives that may occur are resolved locally at each node (as discussed in Chapter 6), thus relieving the network of significant messaging overhead.

5.1.3 The Notification Protocol

When a message FWDRESOURCE containing a publication p of a resource arrives at a node P, the queries matching p are found by utilising its local index structures and using the algorithms described in detail in Chapter 6.

Once all the matching queries have been retrieved from the database, P creates notification messages of the form QNOTIFICATION$(l(r))$ and contacts all the nodes that their queries where matched against p using their IP address associated with the query they submitted. If a node P' is not online when P tries to notify it about

the published resource, the notification message is sent to the $successor(P')$. In this way P' will be notified the next time it logs on the network. To utilise the network in a more efficient way, notifications can also be batched and sent to the subscribers when traffic is expected to be low.

5.1.4 Frequency Cache

In this section we introduce an additional routing table that is maintained in each node. This table, called *frequency cache (FCache)*, is used to reduce the cost of publishing a resource. Using the protocols described earlier, each node is responsible for handling queries that contain a specific word. When a resource r with h distinct words is published by node P, P needs to contact at most h other nodes which will match the incoming resource against their local query databases. This procedure costs $O(h \log N)$ messages for each resource published at P. Since some of the words will be used more often at published resources, it is useful to store the IP addresses of the nodes that are responsible for queries containing these words. This allows P to reach in a single hop the nodes that are contacted more often.

FCache is a hash table used to associate each word that appears in a published document with a node IP address. It uses a word w as a key, and each FCache entry is a data structure that holds an IP address. Thus, whenever P needs to contact another node P' that is responsible for queries containing w, it searches its FCache. If FCache contains an entry for w, P can directly contact P' using the IP stored in its FCache. If w is not contained in FCache, P uses the standard DHT lookup protocol to locate P' and stores contact information in FCache for further reference. Using FCache, the cost of processing a published resource p is reduced to $O(v+(h-v)\log N)$, where v is the number of words of p contained in FCache. Notice that the construction and maintenance of FCache comes at no extra message cost and node routing information is discovered only when needed. In the experiments presented in the next section we discuss good choices for FCache size (see Section 5.2.2).

The extra cost involved with FCache is possible cache misses because of network

dynamicity. In an FCache miss, the node needs to utilise the routing infrastructure at the cost of $O(logN)$ messages to locate a node. However, the new contact information is used to update the FCache entry for future reference. Misses are most likely to occur for infrequent words, since nodes responsible for storing queries with frequent words will be contacted repeatedly.

5.2 Experimental Evaluation

To carry out the experimental evaluation of the protocols described in the previous section, we needed metadata for incoming resources, as well as user queries. For the model $\mathcal{AWPS}$ considered in this work there are various document sources that one could consider: TREC corpora, metadata for papers on various publisher Web sites (e.g., ACM or IEEE), electronic newspaper articles, articles from news alerts on the Web (e.g., `http://www.cnn.com/EMAIL`) etc. However, it is rather difficult to find user queries except by obtaining proprietary data (e.g., from CNN's news or Springer's journal alert system).

For our experiments we use 10426 documents downloaded from CiteSeer[1] and used also in [74, 195, 200, 202, 203]. The documents are research papers in the area of Neural Networks and we will refer to them as the NN corpus[2]. Because no database of queries was available to us, our queries are synthetically generated by exploiting 2000 documents of the corpus. The remaining 8426 documents are used to generate publications.

Each query q has two parts: (i) a Boolean part which consists of at most 4 conjuncts that are atomic Boolean queries of the form $A \sqsupseteq wp$, where wp is a conjunction of at most 4 words or proximity formulas, and (ii) a VSM part which consists of at most 3 conjuncts of the form $A \sim_k s$, where s is a text value. Each atomic Boolean query of the form $A \sqsupseteq wp$ is generated using the methodology of [195, 200, 202, 203]. We set A to be TITLE, AUTHORS, ABSTRACT or BODY with some probability. Then, we set wp to a conjunction of words or proximity formulas

[1] `http://citeseer.ist.psu.edu`

[2] We would like to thank Evangelos Milios and his group at Dalhousie University for providing us the original Neural Network Corpus.

obtained from technical terms mined from the document corpus. Each atomic VSM query of the form $A \sim_k s$ is generated as follows. We set A to be TITLE, ABSTRACT of BODY with some probability. Then, we choose randomly a corpus document and set s equal to the TITLE, ABSTRACT or some part of the BODY field depending of our earlier choice of A. Finally, we set k to a value between $[0.3, 0.7]$ using the uniform distribution.

We have implemented and experimented with six variations of the DHTrie protocols. The first one, named It, utilises the iterative method in the publication protocol and does not use FCache. The second algorithm, named ItC, utilises again the iterative method and also an FCache, and is intended to show the effect of FCache when using the iterative method in the publication protocol. The third algorithm, named Re, utilises the recursive method in the publication protocol but does not use the FCache. ReC uses the recursive method and FCache and shows a significant improvement regarding network utilisation compared to the rest of the algorithms. Finally, the algorithms Hy and HyC use the hybrid method described in Section 5.1 and target low latency in the publication of a document and in the indexing of a vector space query. Algorithm Hy does not utilise an FCache, whereas HyC does. All the algorithms and the DHTrie simulator were implemented in C/C++.

To carry out each experiment described in this section, we execute the following steps. Initially the network is set up by assigning keys to nodes. These keys are calculated using the SHA-1 cryptographic hash function and randomly created IP addresses and ports. After the network is set up, we create 5 million user queries and distribute them among the nodes using the protocol described in Section 5.1.1.

In the experiments described in this section, we are mainly interested in the performance of the six different algorithms in terms of network traffic and latency to publish a document. To measure network traffic, we publish the corpus documents at different nodes and record the network activity. According to the publication protocol, the number of posted queries does not affect the cost for publishing a document in the network. It only affects the matching time for the local filtering algorithms and the number of matching notifications produced (the higher the number of posted queries, the higher the number of matching notifications produced).

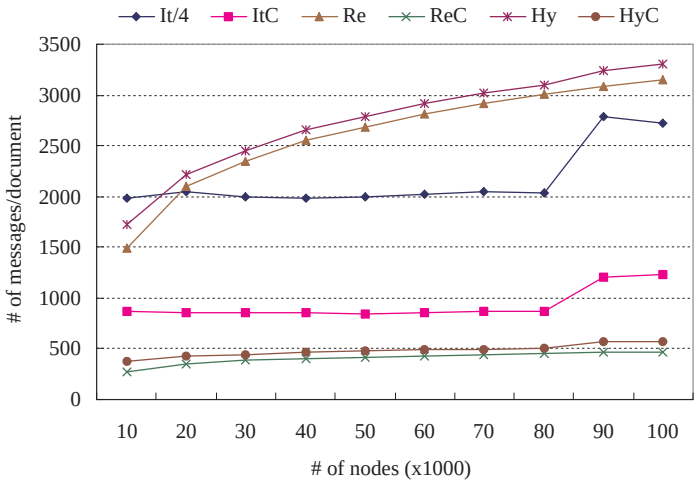
Parameter	Description
N	# of nodes in the system
Q	# of queries assigned to nodes
C_s	# of entries in FCache
C_t	# of publications used to train FCache
W	average # of words per published document
SF	split factor (used for load balancing)
T	split threshold (used for load balancing)
S	size of recipients list (used in hybrid method)
δ	tolerance factor (used in hybrid method)

Table 5.1: Parameters varied in experiments and their descriptions

Publication latency is measured in number of messages as follows. For each document published, we record the longest chain of messages needed until the publication reaches all the intended recipients. As we will show in the experiments, algorithms It and ItC have the lowest latency since all publication messages are sent in parallel (but with high cost in network traffic). On the other hand, algorithms Re and ReC show the worst performance in publication latency due to the recursive way they use to contact the responsible nodes (although we will show that network traffic is significantly reduced). Finally, algorithms Hy and HyC are a tradeoff between the previous algorithms, trying to reduce latency (by parallelising messages up to a certain extent) but at the same time keep the message traffic low. Table 5.1 summarises the parameters used in the experiments. Next to each graph we show later, a table containing the baseline values for these parameters is provided.

5.2.1 Varying Network Size

The first set of experiments that we conducted to evaluate our protocols targeted the performance of the algorithms in terms of message traffic and publication latency for different network sizes. In this experiment, we randomly selected 100 documents (with 5415 words average size) from the NN corpus and used them as incoming

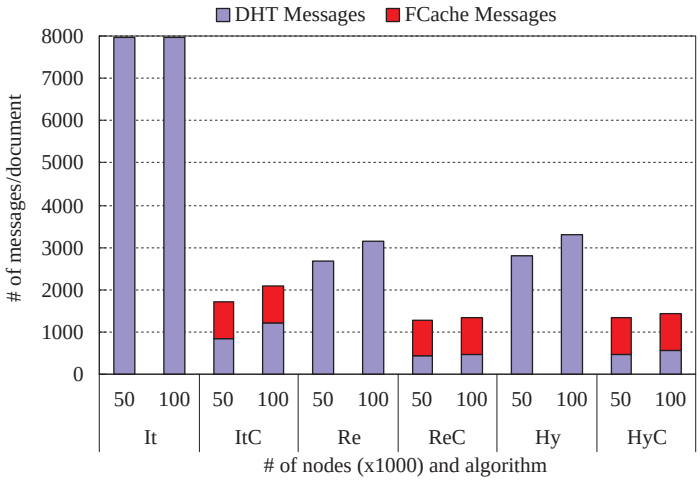


Parameter	Value
N	10K-100K
Q	5M
C_s	30K
C_t	10K
W	5415
SF	-
T	-
S	50
δ	5

Figure 5.5: Performance in terms of message traffic for various network sizes

publications by randomly assigning each one to a publisher node. In each one of the 10 different runs, each document was assigned to a different node. Having published the documents, we recorded the total number of DHTrie messages generated by the network in order to match these documents against the posted user queries.

In Figure 5.5, the performance of the different algorithms in terms of DHTrie messages per document is shown. To improve the readability of the graph and show clearer the behaviour of the algorithms performing better, the measurements of algorithm It have been subquadrupled. The main observation is that the number of messages generated by all the different algorithms to match the incoming documents against the user queries, grows at a logarithmic scale mainly due to the routing infrastructure used. A second observation emerging from the graphs is the effectiveness of the FCache independently of the message routing algorithm used. The use of FCache (with 30K entries in this experiment) results in the reduction of messages sent using the routing infrastructure by more than 6 times in the recursive and the hybrid method, and by 8 times in the iterative method. Notice also that using algorithm ReC reduces the DHTrie message cost of publishing a document of about 5500 words to only 500 messages for a network consisting of 100K nodes managing to process both Boolean and VSM queries. Finally, it is worth pointing out the small difference in the performance of the recursive and the hybrid methods, with the recursive one being slightly better. This is important since as we will show



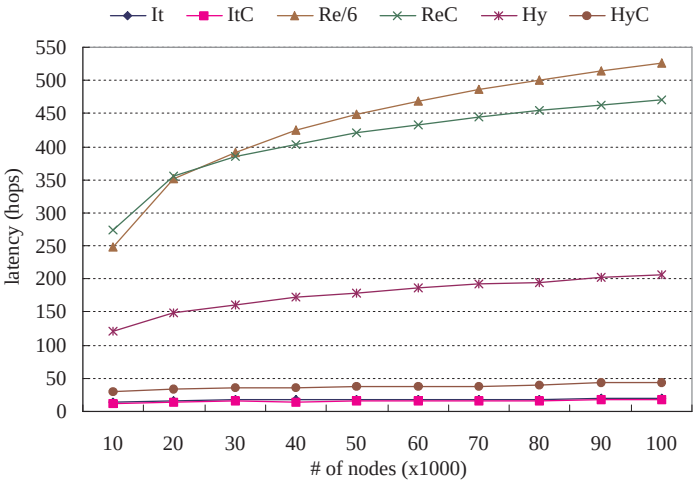
Parameter	Value
N	50K, 100K
Q	5M
C_s	30K
C_t	10K
W	5415
SF	-
T	-
S	50
δ	5

Figure 5.6: Total document processing cost

later in this section the hybrid method is much more efficient in terms of publication latency, a fact that makes it an appealing approach.

In Figure 5.6, we present the total cost for processing a single document in terms of message traffic for networks of 50K and 100K nodes. By total cost we mean the messages sent using information from FCache plus the messages sent using the DHTrie infrastructure. For readability reasons, messages for algorithm It have been truncated to 8000. For algorithm ItC and a network of 50K nodes, the DHTrie messages were about 50% of the total messages sent, whereas for a network of 100K nodes they were about 60%. On the other hand, for algorithms ReC and HyC the DHTrie messages were around 35% for both network sizes. The above observations show the importance of these two methods and FCache in the reduction of the total document matching cost and in the relief in terms of messages of the DHTrie routing infrastructure.

Finally in Figure 5.7, we show the performance of the six algorithms in terms of publication latency and how this performance is affected by increasing the number of nodes in the network. For readability reasons, we have reduced the latency measurements of algorithm Re by a factor of 6. One important observation is the good performance of the iterative algorithms in terms of latency. This is due to Chord's lookup protocol used. A single lookup message is sent to each one of the intended recipients of the publication, and thus the publication process is



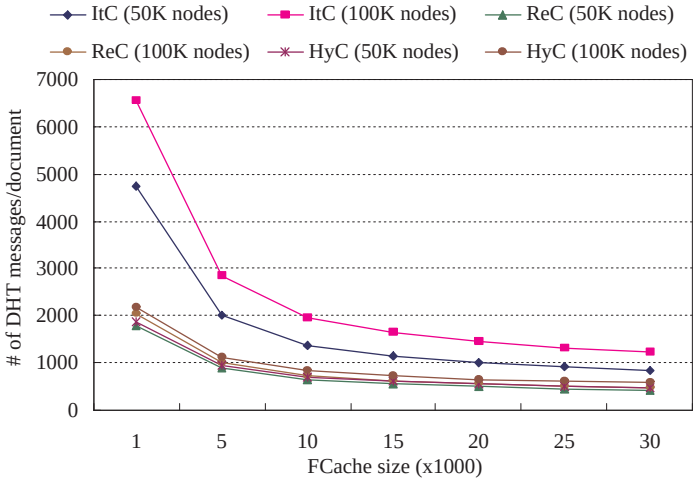
Parameter	Value
N	10K-100K
Q	5M
C_s	30K
C_t	10K
W	5415
SF	-
T	-
S	50
δ	5

Figure 5.7: Performance in terms of publication latency for various network sizes

parallelised. In this way, the longest chain of messages created is that imposed by Chord's routing strategy, which is logarithmic to the number of nodes in the system and independent of the number of recipients. On the other extreme are the recursive methods, that perform poorly. This is due to the long recipients list used to contact the intended recipients at publication time, as described in Section 5.1.2. Finally, the hybrid methods present a better behaviour to publication latency. An important observation that should be stressed is the use of the FCache and its effect in the reduction of publication latency. Algorithm ItC is for example 6 times faster than It and the performance of HyC is comparable to that of the iterative algorithms. The utilisation of the FCache manages to remove many recipients (by contacting them directly using their IP stored in the FCache) from recipients lists created by the recursive and hybrid methods, thus significantly reducing publication time. This also explains the reason for not improving the performance of the iterative method.

5.2.2 Varying the FCache Size

The second set of experiments targeted the performance of the algorithms under different FCache sizes and studied the effect of FCache in the DHTrie utilisation and also in publication latency. We used the document corpus as the training set for populating the FCache of the different nodes. A randomly chosen node p publishes 10K documents and populates of its FCache with the IP addresses of the

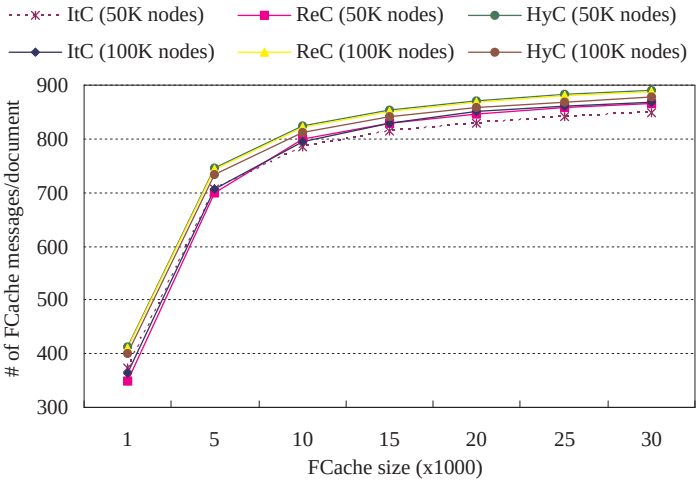


Parameter	Value
N	50K, 100K
Q	5M
C_s	1K-30K
C_t	10K
W	5415
SF	-
T	-
S	50
δ	5

Figure 5.8: Message traffic at the DHT for different FCache sizes

nodes that are responsible for the most frequent words contained in the published documents. These publications served as a training set for the FCache. Then another 100 documents are published by p and the size of the FCache is limited to different values. Subsequently, the total number of messages, used to match these documents against the stored user queries, is recorded. Figure 5.8 shows the utilisation of the overlay network in messages per document, as the size of FCache grows. The values shown are averaged over 10 runs with different nodes.

As it is shown in Figure 5.8, the number of messages sent using the DHTrie routing infrastructure reduces quickly as the size of FCache increases to reach a state where the effect of an FCache increase causes no significant change in the number of messages (around 30K entries, the rightmost point in x-axis). Notice that the cost for each node to maintain an FCache consists only of storing some information in its local data store, namely about 24 bytes per entry (the hash value of the word and the IP address of the node responsible for this word). Additionally, the routing information of the FCache of node p depends only on the documents that get published by p, causing no additional maintenance messages. The only extra cost involved with FCache is its update cost as nodes come and go from the network. This causes FCache entries to be outdated, costing more extra messages through the routing infrastructure to publish a resource. These extra messages though are sent only once, since the FCache field is updated when the new node responsible for



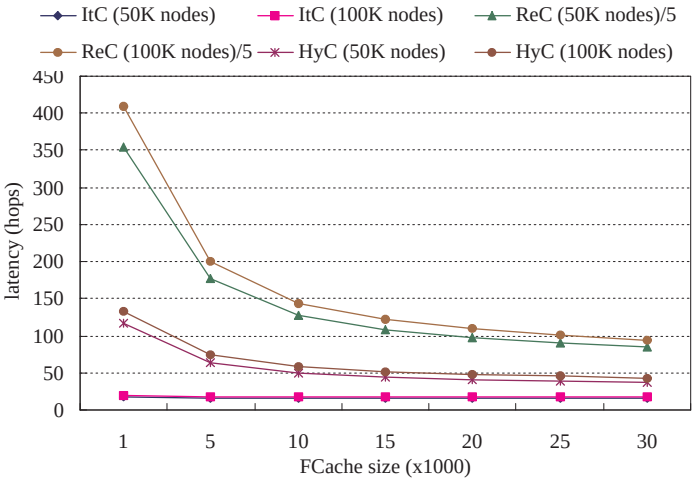
Parameter	Value
N	50K, 100K
Q	5M
C_s	1K-30K
C_t	10K
W	5415
SF	-
T	-
S	50
δ	5

Figure 5.9: Number of messages sent by utilising the FCache, for different FCache sizes

the word with an outdated entry is located. With our measurements, we found out that when 10% of the FCache entries are outdated, the message cost increase was no more than 4% showing that FCache is able to cope up with misses. Notice also that in the recursive and hybrid methods (algorithms ReC and HyC) the performance of FCache in different network sizes remains constant, whereas in the case of the iterative method (algorithm ItC) the performance deteriorates (we get 50% more DHTrie messages per document for an 100% increase in network size).

Figure 5.9 shows the utilisation of FCache per document, showing again that after a threshold value (in our example around 30K entries, the rightmost point in x-axis) its effect is significantly reduced. This is also the reason that we chose 30K FCache entries as a baseline value for the rest of our experiments. We also observe that the number of messages sent using the FCache is about the same for all the algorithms and network sizes, showing that FCache is equally utilised in all cases.

In Figure 5.10 we show the performance of the algorithms in terms of publication latency and how this performance is affected by the variation of the FCache size. An important observation emerging from the graphs is that the effect of the FCache size is not the same for all the algorithms. ItC remains unaffected by the increase not only in FCache size but also in the network size, something that is also verified from the graphs of the previous section. This is due to the routing infrastructure and the iterative way of publishing the incoming documents. On the other hand



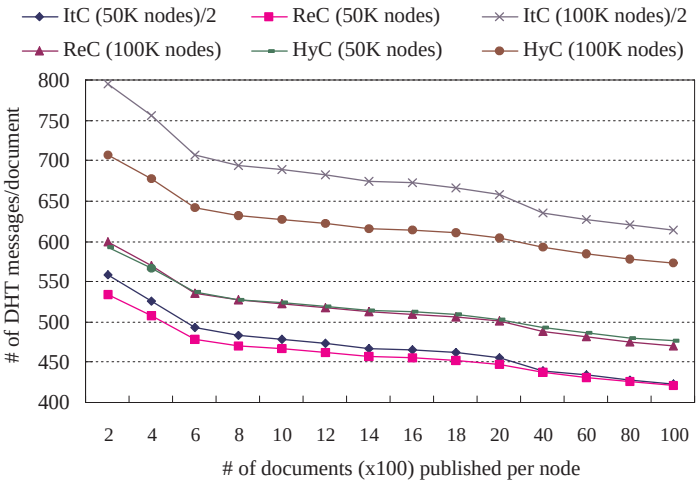
Parameter	Value
N	50K, 100K
Q	5M
C_s	1K-30K
C_t	10K
W	5415
SF	-
T	-
S	50
δ	5

Figure 5.10: Publication latency for different FCache sizes

algorithms *ReC* and *HyC* seem to perform better when the size of the FCache increases. This is attributed, as also stated in the previous section, to the removal of many recipients from recipients lists that results in smaller message chains, and thus smaller latency. As the size of FCache increases, more recipients are contacted using their IP (obtained from FCache) rather than the routing infrastructure, which causes shorter publication times. Finally, although a slight increase due to network size is observed, the behaviour of the algorithms and the FCache effect remain unaffected by this change.

5.2.3 Effect of FCache Training

In this set of experiments we measure the effect of FCache training to the message cost imposed to the network by the publication of a single document and also to the publication latency. To conduct the experiments, we randomly selected a node P and trained P's FCache with a varying number of documents. Through this process the node was able to collect statistics about the most frequent words used in documents (published by it), and as a result it was able to populate its FCache with the appropriate pointers to frequently contacted nodes. Thus, for an FCache with 30K entries (the baseline value used in the experiments), the node would know the IP addresses for the nodes responsible for the 30K most frequent words. We then published 100 documents (with 5415 words average document size) to P and



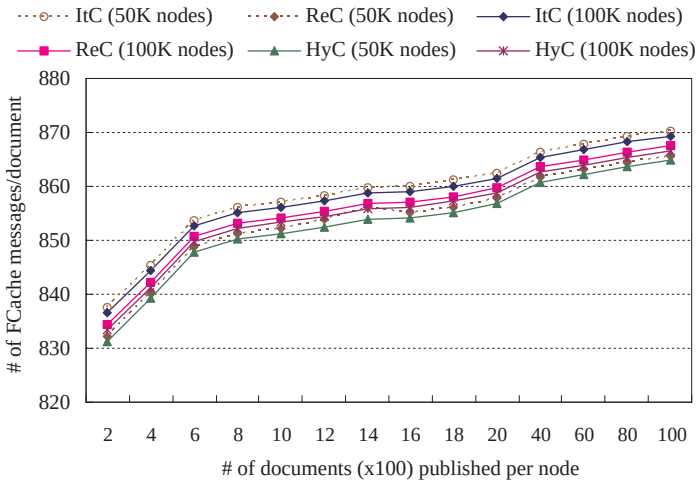
Parameter	Value
N	50K, 100K
Q	5M
C_s	30K
C_t	0.2K-10K
W	5415
SF	-
T	-
S	50
δ	5

Figure 5.11: Performance of the DHT for different levels of FCache training

recorded the message cost to match these documents against the stored user queries. The results shown in Figure 5.11 are averaged over 100 runs for different nodes to eliminate network topology effects.

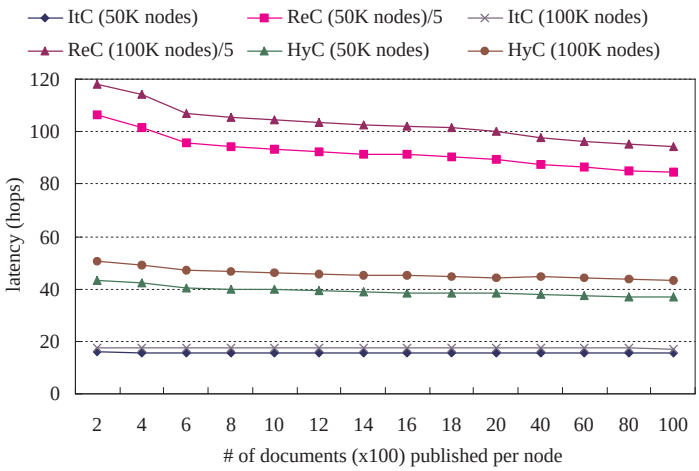
Figure 5.11 shows that the performance of the different algorithms improves as more documents get published. For readability purposes we have reduced the measurements for the ItC algorithm by a factor of 2. Algorithms ReC and HyC seem less sensitive in this parameter, as the difference in the number of messages observed is about 100 messages for 50 times more documents (the leftmost and rightmost point in the x-axis), whereas ItC presents a difference of more than 300 messages. Additionally, ReC shows less sensitivity with respect to the network size, contrary to ItC that needs about 50% more messages. Finally, all algorithms show a similar behaviour for the two network sizes we tested.

Figure 5.12 shows the number of hits of FCache for different levels of FCache training. Notice that all the algorithms have roughly the same number of hits for a network of 50K nodes, showing that FCache hits are not affected by the algorithm used. Looking at the scale in the y-axis, we can also see that the number of FCache hits shows only a slight improvement of around 4% for a 5000% increase in the number of documents used for training. This is attributed to the skewed nature of the data (documents) used to train the FCache. It is however important to note that even a small increase in FCache hits can significantly reduce message load (as



Parameter	Value
N	50K, 100K
Q	5M
C_s	30K
C_t	0.2K-10K
W	5415
SF	-
T	-
S	50
δ	5

Figure 5.12: Number of messages sent by utilising the FCache, for different levels of FCache training

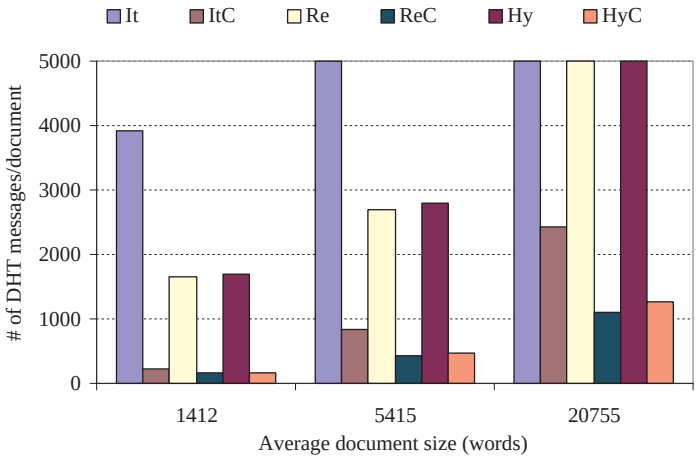


Parameter	Value
N	50K, 100K
Q	5M
C_s	30K
C_t	0.2K-10K
W	5415
SF	-
T	-
S	50
δ	5

Figure 5.13: Publication latency for different levels of FCache training

it is already shown in the graphs of Figure 5.12), since every FCache hit, saves us $O(\log N)$ DHTrie messages.

In Figure 5.13 we show the performance of the algorithms in terms of publication latency and how this performance is affected as FCache training varies. We observe the algorithm *ReC* is the most heavily affected by the training level of the FCache, followed by *HyC* and *ItC* which remains unaffected by the variation in the FCache training. This can be explained as follows. Regarding publication latency, *ReC* is the algorithm that is mostly dependent on the information provided by FCache, as this reduces the long recipients lists that delay the document publication. Algorithm *HyC* is designed to produce shorter lists and thus the removal of many recipients is



Parameter	Value
N	50K
Q	5M
C_s	30K
C_t	10K
W	1412, 5415, 20755
SF	-
T	-
S	50
δ	5

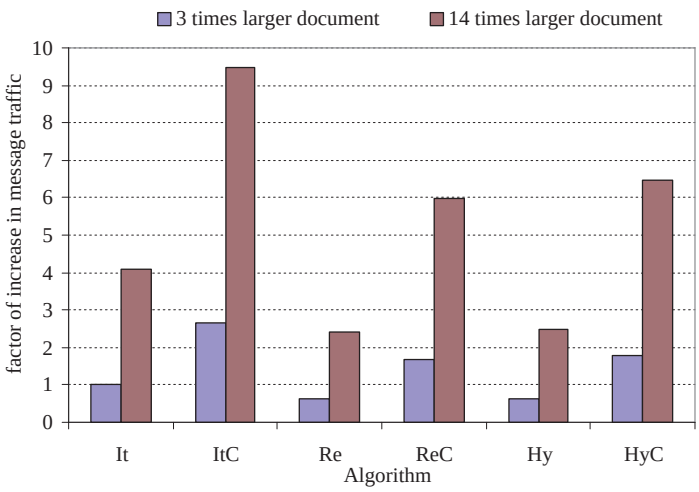
Figure 5.14: Message traffic at the DHT for documents of different size

not crucial for its performance in publication time. Algorithm ItC uses the iterative way provided by Chord infrastructure to contact message recipients, thus remains unaffected by FCache training. Finally all algorithms show a slight increase when the network size is doubled. This increase is small due to the logarithmic routing provided by the infrastructure.

5.2.4 Varying the Document Size

Document (i.e., publication) size is an important parameter in the performance of our algorithms. This set of experiments targeted the performance of the different algorithms for varying document sizes. Figure 5.14 shows the message cost for publishing documents of varying size by using each one of the six different algorithms. Each one of the bars in Figure 5.14 is an average of the message cost for 100 documents, published by 1000 different nodes (in a network of 50K nodes in total) to normalise network topology effects. Notice that the graph is truncated to a maximum of 5000 messages to show clearly the best performing algorithms.

Figure 5.14 shows that for small documents the use of the recursive or the hybrid method, contrary to FCache, does not improve performance significantly, since algorithms ItC, ReC and HyC perform similarly. This is because for a large proportion of the words contained in small documents an FCache entry exists, thus needing a single message to reach the node responsible for queries that contain these words.

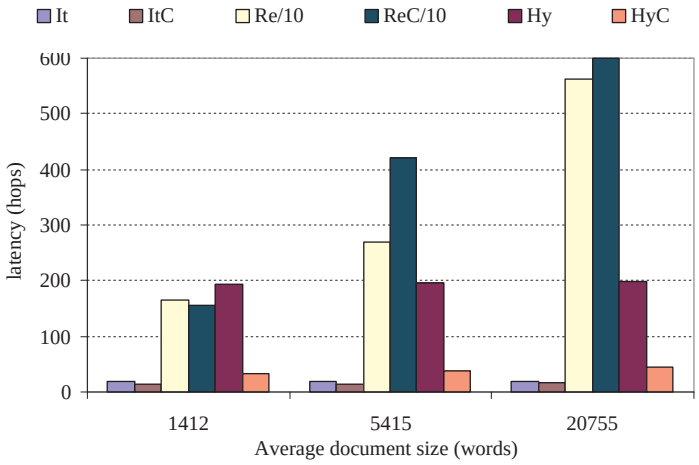


Parameter	Value
N	50K
Q	5M
C_s	30K
C_t	10K
W	1412, 5415, 20755
SF	-
T	-
S	50
δ	5

Figure 5.15: Increase rate in message traffic with respect to document size for each algorithm

The remaining words that are not listed in node's FCache use the DHTrie infrastructure, but their number is so small that we cannot observe significant differences in message cost. For large documents though, the use of the recursive method together with FCache is shown to be significantly better than its counterparts, managing to process documents of average size of around 21K words by using around 1800 messages. Note also that algorithms ReC and HyC perform similarly for all sizes of documents, underpinning our initial claim that the two algorithms behave similarly in terms of message traffic, but HyC is a better choice since it handles publication latency in a more efficient way.

For this experiment we used three groups of 100 documents, D_1, D_2 and D_3. Document group D_2 was 3 times larger on average than D_1, while document group D_3 was 14 times larger on average than D_1. Initially 100 random nodes were chosen to publish document group D_1, and the message traffic generated was recorded. Then in a similar way the other two document groups were published and the number of messages generated was recorded. Figure 5.15 shows the factor of increase in message traffic for each algorithm when publishing the two different groups of documents. With our experiments, we have concluded that the increase in message cost is linear to the document size for all algorithms with algorithm Re and Hy presenting the smaller increase factor, thus showing a smaller sensitivity to document size. On the other hand, algorithms that use the FCache show more sensitivity



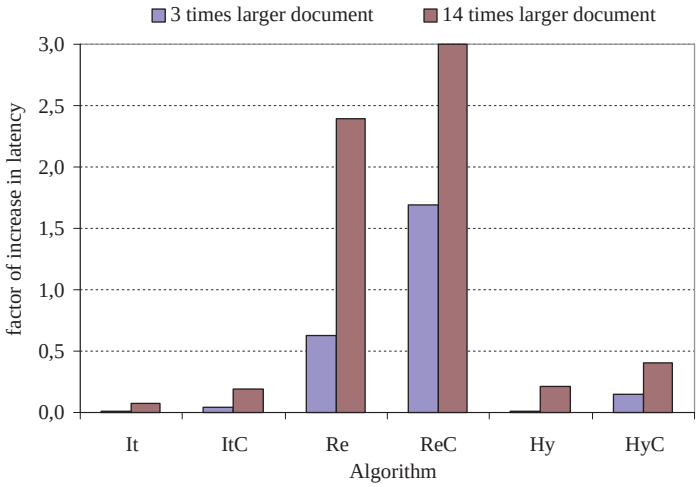
Parameter	Value
N	50K
Q	5M
C_s	30K
C_t	10K
W	1412, 5415, 20755
SF	-
T	-
S	50
δ	5

Figure 5.16: Latency for documents of different size

to document size, with HyC being the algorithm with the second larger increase rate. Despite this increase, HyC remains the most efficient algorithm (together with ReC) as shown in Figure 5.14.

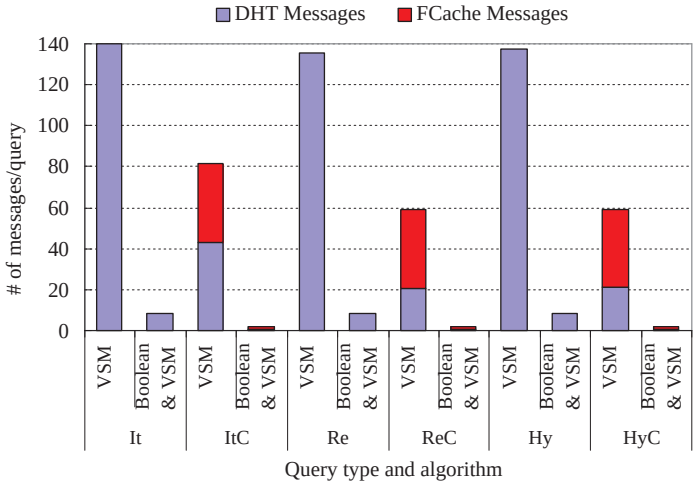
Figure 5.16 shows how document size affects latency for the different algorithms. The most important observation is the bad behaviour of the recursive algorithms (notice that the measurements for these algorithms are reduced by a factor of 10 for better readability), which shows that they are heavily affected by the document size. This is expected since the recipients list grow longer with publication size, making it a crucial parameter for the performance of the recursive algorithms. On the other hand, the iterative and hybrid algorithms are not sensitive to document size, for different reasons each. The iterative algorithms are not sensitive because of the usage of the Chord lookup functionality (and no recipients lists) and the hybrid methods because of the use of recipients lists that are independent from the publication size.

Similarly to Figure 5.15, the graphs in Figure 5.17 show the factor of increase in publication latency for each algorithm when publishing a 3 times larger and a 14 times larger document. We observe that the iterative and hybrid algorithms are almost insensitive to document size, while the recursive algorithms show high sensitivity. This is caused by the recipients lists which grow longer with publication size.



Parameter	Value
N	50K
Q	5M
C_s	30K
C_t	10K
W	5415, 20755
SF	-
T	-
S	50
δ	5

Figure 5.17: Increase rate in latency with respect to document size for each algorithm

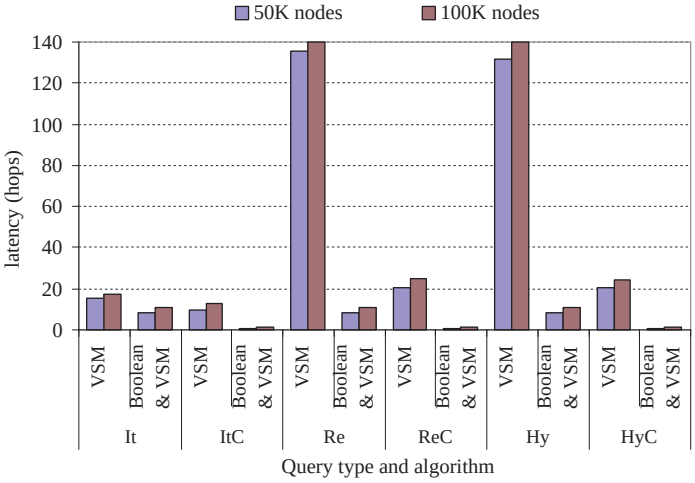


Parameter	Value
N	50K
Q	500K
C_s	30K
C_t	10K
W	-
SF	-
T	-
S	50
δ	5

Figure 5.18: Message cost to index a query in the network

5.2.5 Varying the Type of Queries

In this set of experiments we investigate the cost of indexing a query in the network. Each one of the bars in Figures 5.18 and 5.19 shows the average message traffic and latency recorded when indexing 500K queries of each type in the network of 50K nodes. In these experiments we used queries with vector space atomic queries only and also queries with both Boolean and vector space queries. Indexing the second type of queries is the same as indexing queries with Boolean atomic queries only (the user is referred to Section 5.1.1). Figure 5.18 shows the message cost for indexing queries of different types by using each one of the six different algorithms. Notice that the graph is truncated to a maximum of 140 messages to show clearly the best



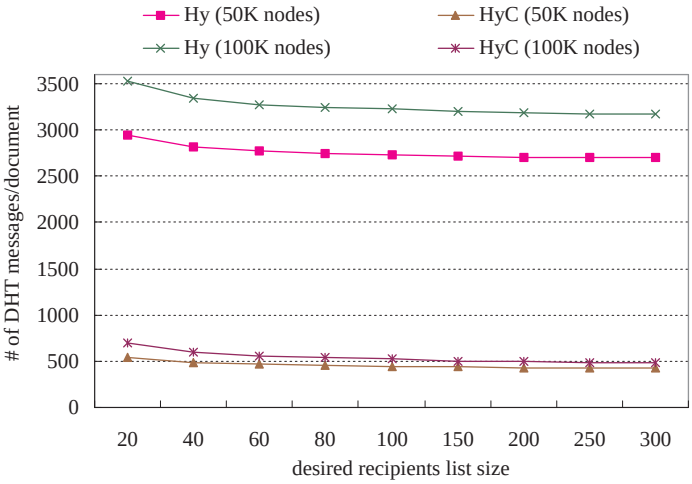
Parameter	Value
N	50K
Q	500K
C_s	30K
C_t	10K
W	-
SF	-
T	-
S	50
δ	5

Figure 5.19: Latency in indexing a query in the network

performing algorithms.

The most important observation in this graph is that regardless of the algorithm, vector space queries are much more expensive to index than Boolean or mixed type queries, since vector space queries require indexing in all nodes responsible for the distinct words contained in the query, whereas other query types are indexed under only one (see Section 5.1.1). It is clear that *ReC* and *HyC* are the best performing algorithms for vector space query indexing in terms of message traffic, whereas no significant difference in performance is observed for the other query types. Notice also the important role of FCache, which manages to forward to the intended recipients more than 2/3 of the total network traffic in algorithms *ReC* and *HyC*, thus relieving the DHT infrastructure of substantial messaging effort.

As we can see in Figure 5.19, latency in the indexing of Boolean or mixed type queries is similar for all the algorithms with the *Re* algorithm being slightly worse than the rest. For vector space queries however algorithms *Re* and *Hy* are performing worse than their counter parts. Additionally, algorithm *Hy* seems to behave similarly in terms of latency to *Re* which may seem unusual considering its performance in the previous experiments. This change in behaviour is explained by the fact that in this experiment the two algorithms have roughly the same size of recipients lists. This happens because *Re* creates small recipients lists (since a vector space query in our scenario is about the size of an paper abstract) and thus the size



Parameter	Value
N	50K, 100K
Q	5M
C_s	30K
C_t	10K
W	5415
SF	-
T	-
S	20-300
δ	5

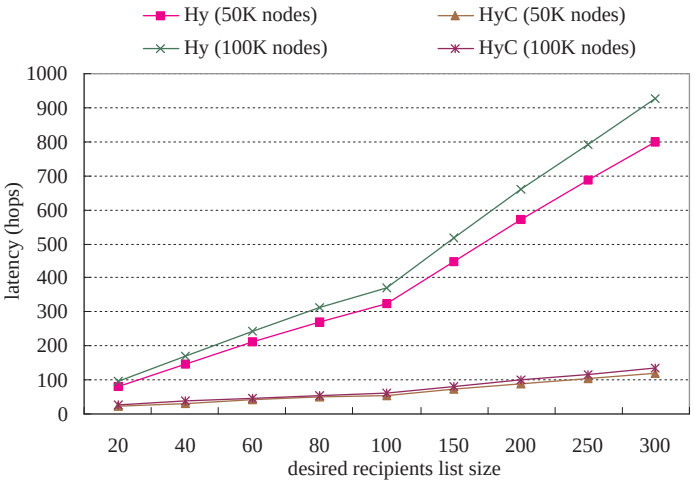
Figure 5.20: Message traffic when varying the recipient list size in the hybrid algorithms

of the list is similar to the size we use for algorithm Hy.

5.2.6 Varying the Desired Recipients List Size in the Hybrid Algorithms

This set of experiments targets a specific parameter of the hybrid algorithms and intends to show how the size of the recipients list affects the message load in the network and publication latency. Figures 5.20 and 5.21 show the Hy and HyC algorithms for two different network sizes (50K and 100K nodes) and each point in the graph is averaged over 10 runs. We have used 100 documents (with 5415 words average size) from the NN corpus as incoming publications and randomly assigned each document to a publisher node.

Figure 5.20 shows the average number of DHT messages needed to publish a document as the desired recipient list size increases. We should note that the hybrid algorithm tries to bridge the two extremes between the iterative and the recursive algorithms. The lower the recipients list size the closer the algorithm is to the iterative counterpart, as the iterative counterpart can be viewed as a hybrid algorithm with recipient list size of one. On the other hand, the higher the recipient list size, the closer the algorithm is to the recursive counterpart. For this reason, we see the reduction in the network traffic as the recipient list size increases. Another impor-



Parameter	Value
N	50K, 100K
Q	5M
C_s	30K
C_t	10K
W	5415
SF	-
T	-
S	20-300
δ	5

Figure 5.21: Latency when varying the recipient list size in the hybrid algorithms

tant observation is the role of the FCache in the behaviour of the algorithms. It decreases network traffic by a factor of 6 as it was expected given the rest of results presented in this section, and reduces the effect of the network size in the behaviour of the algorithm (notice for example the difference between the two instances of algorithm Hy and the difference between the two instances of algorithm HyC).

When publication latency is in question then the behaviour of the algorithms is similar (Figure 5.21). FCache continues to play an important role in reducing latency and network size effects, while all algorithms show an increase when the desired recipients list size increases. This behaviour can be explained as above by the tuning of the hybrid algorithm between the two extremes.

5.2.7 Summing Up

We have presented a set of protocols, collectively called *DHTrie*, that extend the Chord protocols with pub/sub functionality. We have implemented six variations of the protocols and evaluated them experimentally in different settings and for different parameters. In this section we present our conclusions regarding the experimental evaluation of the six algorithms and sum up the strengths and weaknesses of each one of them.

Experiments have shown the usefulness the FCache in all the cases. In all our experiments the algorithms that use the FCache (ItC, ReC and HyC) performed

significantly better than their counterparts (It, Re and Hy), not only in terms of message traffic but also in terms of publication latency. This, combined with the fact that FCache is a local data structure that is compact (less than 1MB) and easy to maintain, lead us to the conclusion that such a routing table is a useful tool that comes with very little extra cost.

When message traffic in the routing infrastructure is in question, then algorithm ReC is the best candidate. It remains relatively unaffected by network size, it shows relatively small sensitivity to increasing document sizes and needs only few valid FCache entries to present a big performance improvement. On the other hand, when publication latency is in question, algorithm ItC is a good candidate. Its performance with respect to publication latency remains unaffected by network size, level of training and size of the FCache, and the size of the publication. This of course comes at the price of higher network traffic in the routing infrastructure. Finally, HyC is a tunable alternative to the previous approaches that manages to balance its performance between network traffic and publication latency. It is slightly more expensive than ReC in terms of network traffic but still significantly lower than ItC, whereas it performs well in terms of latency. Its behaviour shows no deviations from the behaviour of the other two algorithms and its sensitivity to different parameters follows that of its counterparts. In general it tries to combine the benefits of the two previous algorithms by providing an versatile algorithm that can be tuned to optimise either network traffic or publication latency. HyC should be the algorithm of choice when our performance metric gives equal weight to both message traffic and latency, and should be a reasonable choice for a distributed resource sharing scenario, as the one we have assumed.

5.3 Load Balancing

In typical IR scenarios the probability distributions associated with documents and queries can be arbitrary and are typically skewed. For example, the frequency of occurrence of words in a document collection follows the Zipf distribution, subscriptions to an electronic journal might refer mostly to current hot topics while publications appearing in the same journal might reflect its established tradition

etc. Thus, a key issue that arises when trying to partition the query space among the different nodes of a DHT in a pub/sub scenario is to achieve *load balancing*. In any pub/sub setting we can distinguish three types of node load: *query*, *routing* and *filtering*.

The *query load* of a node P is the number of queries stored at node P. The *routing load* of a node P is the number of messages that P has to forward due to the DHTrie protocols. Finally, the *filtering load* of a node P is the number of filtering requests (i.e., publications) that need to be processed at node P.

5.3.1 Balancing the Filtering Load

Filtering is arguably the heaviest of the load balancing tasks at hand, since for each filtering request, a node has to search its local data store, retrieve the matching queries and notify interested subscribers. The filtering load of a node P depends on the number of words that hash to the interval of the identifier space owned by P and the frequency distribution of these words in published documents.

In the DHT literature, work on load balancing has recently concentrated on two particular problems: *address-space load balancing* and *item load balancing*. The former problem is how to partition the address-space of a DHT "evenly" among keys; it is typically solved by relying on consistent hashing and constructions such as virtual servers [180] or potential nodes [113]. In the latter problem, we have to balance load in the presence of data items with arbitrary load distributions [4, 7, 113] as in our case.

We have implemented and evaluated a simple algorithm for distributing the filtering load evenly throughout the different nodes of the network. The algorithm is based on the well-known concept of *load-shedding* (LS), where an overloaded node attempts to off-load work (i.e., filtering requests) to less loaded nodes. The algorithm is in fact applicable to the standard DHT look-up problem but here we utilize it in a pub/sub scenario.

The load balancing algorithm is as follows. Once a node P understands that it has become overloaded, it chooses the most frequent word w it is responsible

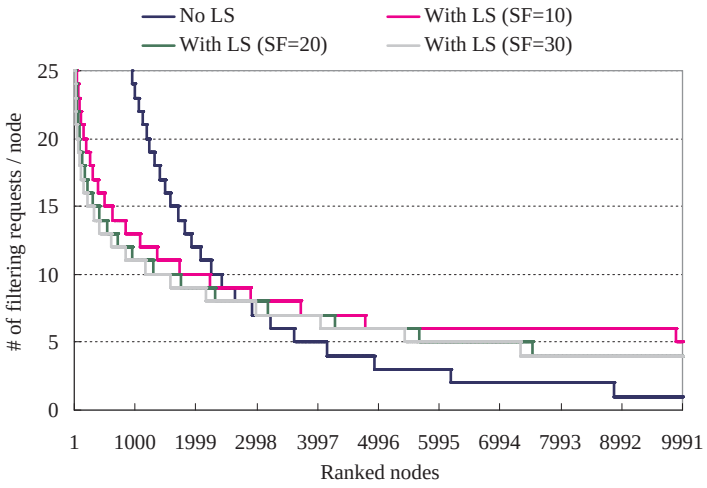
for and a small integer k. Then P contacts the nodes responsible for words wj for all j, $1 \leq j \leq k$ (wj is the concatenation of strings w and j) and asks them to be its replicas[3]. Then P notifies the rest of the network about this change in responsibilities by piggy-backing the necessary information in DHTrie maintenance messages.

Each node M that receives this message notes down the word w. Later on, if M has a new publication containing w, it divides the filtering responsibility for w among P and k other nodes by concatenating a random number from 1 to k to the end of the w and using DHTrie to find the node responsible for this word. In this way, the filtering responsibility of w for P is reduced by $k+1$ times (k new nodes plus P). We call $k+1$ the *split factor* (SF) in subsequent experiments.

In the experiments carried out in this section, a node P considers itself overloaded if it exceeds the threshold (T) of 10 filtering requests for the same word w in a time window of 100 document publications (in other words, if at least 10% of the published documents contain w). In a real network, a node would not know how to define such a time window. In this case it could use sampling to estimate the average document publication rate, and thus be able to discover if it is doing more filtering work than other nodes.

The results of our experiments for the LS algorithm are shown in Figure 5.22. Figure 5.22 shows the average number of filtering requests received by each node in a time window for a period of 100 time windows. SF was varied between 10 and 30 nodes and T was set to 10 requests/time window respectively. For readability purposes only the first 10K nodes (out of a total of 50K) are shown and the y-axis is truncated to 25 filtering requests (the highest point in the unbalanced case is 159 filtering requests for a single node). Notice that prior to the load balancing algorithm the first 3K nodes get a very large proportion of the filtering requests, whereas the rest of the network receives very few or no requests at all. On the contrary, after the load balancing algorithm is run, only a small amount of nodes receive more that 20 filtering requests, with the rest of the filtering load being distributed in a more uniform way among the nodes. It is also notable that the variation of the SF did

[3]Currently we select the replica nodes randomly, but peer load or locality criteria could be used.



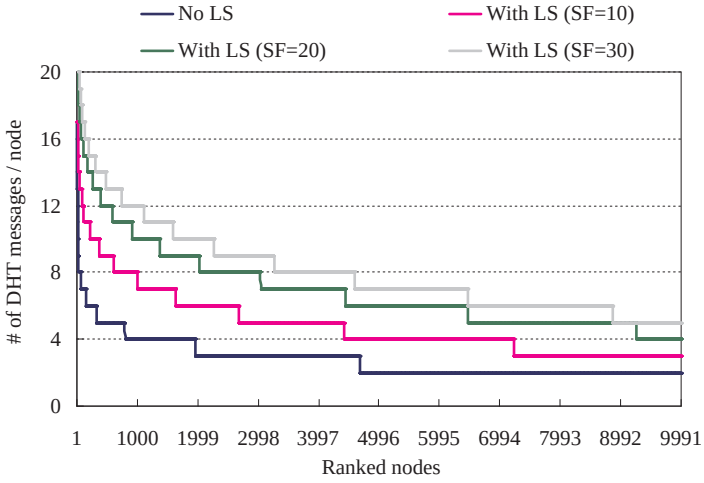
Parameter	Value
N	50K
Q	5M
C_s	30K
C_t	10K
W	5415
SF	1, 10, 20, 30
T	10
S	50
δ	5

Figure 5.22: Average number for filtering requests

not change the balance of the load significantly, although a slight improvement was shown. However, as we will show in the next section, increasing the SF also causes a significant increase in message traffic. Finally, we experimented with different values for T but we did not observe significant differences in the load distribution.

5.3.2 Balancing the Routing Load

Trying to balance the filtering load causes an increase in message traffic. For this reason, we set up an experiment to investigate the effect not only in the amount of traffic added to the system but also the distribution of this addition. Figure 5.23 shows the price we pay to achieve filtering load balancing in terms of message routing. In this graph we show the number of routing requests received by the first 10K nodes of our network. Notice that the number of messages needed per document increases significantly after the load balancing algorithm is run (we observed increases of as much as 80% for SF=10, 180% for SF=20 and 240% for SF=30). For this reason we have used SF=10 wherever the LS algorithm was used. This increase is due to FCache misses occurring from the splitting of queries and filtering responsibilities. The increase in FCache misses causes a significant increase in DHTrie messages as it was expected (see Section 5.2.3), which is reduced when the FCache entries are updated. The important point however in Figure 5.23 is that the new load imposed on the network is uniformly distributed among the nodes and does not



Parameter	Value
N	50K
Q	5M
C_s	30K
C_t	10K
W	5415
SF	1, 10, 20, 30
T	10
S	50
δ	5

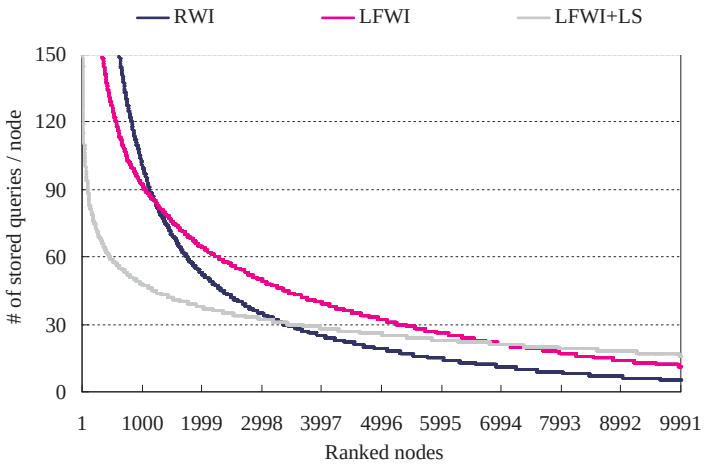
Figure 5.23: Routing load for the first 10K nodes

cause overloading in any group of nodes. The new load distribution follows closely the old one. This observation leads us to the conclusion that the load balancing algorithm shown here manages to efficiently distribute the filtering load among the nodes, while imposing an additional cost for routing purposes, which in our scenario is considered an easier task to perform.

5.3.3 Balancing the Query Load

Even query distribution among nodes is a hard task to achieve since typically queries follow a skewed distribution that reflect hot topics in an area, or popular information needs. Although query load is not one of the heaviest loads imposed on a node, it still remains a significant parameter that has to be addressed. In a distributed resource sharing scenario storage resources or computational power of nodes (e.g., because of battery shortage) may be limited, which makes the cost of storing millions of user queries a difficult one. Additionally, the filtering task is also dawdled when an incoming document has to be matched against millions of queries in a limited resource environment. For the above reasons, we have looked into the problem of query load balancing and we have devised an solution that, combined with the LS algorithm described in Section 5.3.1, manages to distribute the filtering load more evenly to the participating nodes.

Figure 5.24 shows the results for 1M queries indexed in a network of 50K nodes



Parameter	Value
N	50K
Q	5M
C_s	30K
C_t	10K
W	5415
SF	10
T	10
S	50
δ	5

Figure 5.24: Query load for the first 10K nodes

and each graph is produced as an average over ten runs. For readability reasons only the load of the 10K most loaded nodes is shown and the graphs are truncated to a maximum of 150 stored queries per node. To distribute the queries to the nodes responsible we used two different query indexing methods based on the protocols presented in Section 5.1.1. RWI (Random Word Index) chooses to index a query q to the node responsible for a word chosen randomly from the query. Contrary, $LFWI$ (Least Frequent Word Index) chooses to index q to the node responsible for the least frequent word contained in the query. Finally $LFWI$+LS indexes queries under their least frequent word and uses the load shedding algorithm with SF=10 to achieve a better load distribution. We observe that the load distribution of the RWI method is highly skewed with the most loaded node storing 3363 queries, while the majority of the load is handled by 10K nodes. $LFWI$ manages to slightly improve the load distribution, but still the most loaded node is forced to store more than 2000 queries. Finally, we can see the LS algorithm, also applicable in this case, manages to balance the load evenly between the nodes. The most loaded node stores 152 queries while the heaviest 10K nodes (20% of total nodes) store about 35% of total load.

5.4 Conclusions

The evaluation of the DHTrie protocols revealed strengths and weaknesses of the different algorithms developed. In our experiments we showed that the DHTrie protocols are *scalable*: the number of messages it takes to publish a document remains almost constant as the network grows. Additionally, we showed that the use of data structures that exploit local knowledge can significantly reduce network traffic (up to a factor of 9), with little overhead in training. Network traffic also presents little sensitivity to document size when FCache is utilised. Finally, Section 5.3 presents a load balancing algorithm that trades message traffic for balance in the peer load.

In the next chapter, we put our attention on the filtering problem faced by each one of the peers, and provide local data structures and indexing algorithms that enable us to solve the filtering problem efficiently for large databases of queries expressed in the model $\mathcal{AWP}$. To support model $\mathcal{AWPS}$, we utilise an algorithm that combines our approach for indexing Boolean queries with algorithm SQI [211] for indexing VSM queries.

Chapter 6

Local Filtering Algorithms

In Chapters 4 and 5 we dealt with the problem of defining the protocols between the peers to provide pub/sub functionality in a distributed environment. In this setting, peers store continuous queries (or profiles) that express long-term user information needs, while others can publish documents to the network. Whenever a document is published, the continuous queries satisfying this document are found and notifications are sent to appropriate users. This chapter deals with the filtering problem that needs to be solved efficiently by each peer: Given a database of continuous queries db and a document d, find all queries $q \in db$ that match d. In this chapter, we present local data structures and indexing algorithms that enable us to solve the filtering problem efficiently for large databases of queries expressed in the model $\mathcal{AWP}$. Thus, we develop and evaluate efficient main-memory algorithms that are able to filter millions of continuous $\mathcal{AWP}$ queries in just a few hundred milliseconds. Our algorithms are the first in the literature that deal with IR models like $\mathcal{AWP}$ supporting Boolean queries, named attributes with values of type text, and word proximity operators. The results presented in this chapter have been published in [106, 125, 200].

The main idea behind our algorithms is to use tries to capture common elements of queries. In this way, clustering of queries is improved and significantly smaller filtering times are achieved. The algorithms closest to ours are the ones employed in the Boolean version of SIFT [209] where documents are free text and queries

are conjunctions of keywords. SIFT has been the inspiration for this work and the results presented in Sections 6.1 and 6.4 extend and improve the results of [209]. In particular, we evaluate experimentally algorithms BF, SWIN and PrefixTrie that are extensions of the algorithms BF, Key and Tree of [209] for the model $\mathcal{AWP}$. We also discuss in detail the new algorithms BestFitTrie and LCWTrie as alternatives to PrefixTrie, and compare them under various experimental settings. Finally, we introduce ReTrie, an extension of the previous algorithms that considers the periodic reorganisation of the database to achieve greater efficiency at filtering time.

The algorithms presented here are also utilised in systems DHTrie (presented in the previous chapter and also in [203]) and LibraRing (presented in Chapter 4 and also in [202]), in the context of information filtering and digital library applications built on top of DHTs. In all of the above distributed systems, these algorithms are used *locally* in each server, while appropriate protocols (presented in detail in the previous chapter) guarantee successful distributed operation.

The rest of the chapter is organised as follows. Section 6.1 presents four main memory algorithms that solve the filtering problem for *conjunctive queries* in $\mathcal{AWP}$, while Section 6.2 presents an algorithm for reorganising the query database to achieve even faster filtering times. Section 6.3 discusses how to support indexing of $\mathcal{AWPS}$ queries by combining our algorithms with known approaches in the literature, and Section 6.4 presents an extensive evaluation of the algorithms using a real document corpus and realistically created query databases. Finally, a brief background on tries is provided in Section 6.5, while Section 6.6 summarises our achievements and concludes the chapter.

6.1 Filtering Algorithms for $\mathcal{AWP}$

In this section we present and evaluate four main memory algorithms that solve the filtering problem for *conjunctive queries* in $\mathcal{AWP}$. Thus our algorithms deal with queries of the form $A_1 = s_1 \ \wedge \ \ldots \ \wedge \ A_n = s_n \ \wedge \ B_1 \sqsupseteq wp_1 \ \wedge \ \ldots \ \wedge \ B_m \sqsupseteq wp_m$, where A_i, B_i are attributes that belong to the attribute universe $\mathcal{A}$, each s_i is a text value and each wp_i is a word pattern containing conjunctions of words and proximity formulas with only words as subformulas. Because our work extends and improves

previous algorithms for SIFT [209], we adopt terminology from SIFT in many cases.

6.1.1 The Algorithm BestFitTrie

BestFitTrie uses two data structures to represent each published document d: the *occurrence table* $OT(d)$ and the *distinct attribute list* $DAL(d)$. $OT(d)$ is a hash table that uses words as keys, and is used for storing all the attributes of the document in which a specific word appears, along with the positions that each word occupies in the attribute text. $DAL(d)$ is a linked list with one element for each distinct attribute of d. The element of $DAL(d)$ for attribute A points to another linked list, the *distinct word list* for A (denoted by $DWL(A)$) which contains all the distinct words that appear in $A(d)$.

To index queries BestFitTrie utilises an array, called the *attribute directory* (AD), that stores pointers to word directories. AD has one element for each distinct attribute in the query database. A *word directory* $WD(B_i)$ is a hash table that provides fast access to roots of *tries* in a *forest* that is used to organize *sets of words* – the set of words in wp_i (denoted by $words(wp_i)$) for each atomic formula $B_i \sqsupseteq wp_i$ in a query. The proximity formulas contained in each wp_i are stored in an array called the *proximity array* (PA). PA stores pointers to trie nodes (words) that are operands in proximity formulas along with the respective proximity intervals for each formula. There is also a hash table, called *equality table* (ET), that indexes all text values s_i that appear in atomic formulas of the form $A_i = s_i$.

When a new query q of the form of Definition 5 arrives, the index structures are populated as follows. For each attribute $A_i, 1 \leq i \leq n$, we hash text value s_i to obtain a slot in ET where we store the value A_i. For each attribute $B_j, 1 \leq j \leq m$, we compute $words(wp_j)$ and insert them in one of the tries with roots indexed by $WD(B_j)$. Finally, we visit PA and store pointers to trie nodes and proximity intervals for the proximity formulas contained in wp_j.

Let us now explain how each word directory $WD(B_j)$ and its forest of tries are organised. The main idea behind this data structure is to store *sets of words* compactly by exploiting their *common elements*. In this way, memory space is

Id	Query $B_i \sqsupseteq wp_i$	Identifying Subsets
0	$B_i \sqsupseteq$ databases	{databases}
1	$B_i \sqsupseteq$ relational $\prec_{[0,2]}$ databases	{databases, relational}
2	$B_i \sqsupseteq$ databases $\wedge$ relational	{databases, relational}
3	$B_i \sqsupseteq$ (software $\prec_{[0,2]}$ neural $\prec_{[0,0]}$ networks) $\wedge$ (software $\prec_{[0,3]}$ relational $\prec_{[0,0]}$ databases)	{databases, relational, neural}, {databases, relational, software}, {databases, relational, networks}
4	$B_i \sqsupseteq$ optimal $\wedge$ (artificial $\prec_{[0,0]}$ intelligence) $\wedge$ relational $\wedge$ databases	{databases, relational, artificial, intelligence, optimal}
5	$B_i \sqsupseteq$ artificial $\wedge$ relational $\wedge$ intelligence $\wedge$ databases $\wedge$ knowledge	{databases, relational, artificial, intelligence, knowledge}

Table 6.1: Identifying subsets of $words(wp_i)$ with respect to $S = \{words(wp_i),\ i = 0, \ldots, 5\}$

preserved and filtering becomes more efficient as we will see below.

Definition 11 *Let S be a set of non-empty sets of words and $s_1, s_2 \in S$ with $s_2 \subseteq s_1$. We say that s_2 is an* identifying subset *of s_1 with respect to S iff $s_2 = s_1$ or $\nexists\ r \in S$ such that $s_2 \subseteq r$.*

The sets of identifying subsets of two sets of words s_1 and s_2 with respect to a set S is the same if and only if s_1 is identical to s_2. Table 6.1 shows some examples that clarify these concepts.

The sets of words $words(wp_i)$ are organised in the word directory $WD(B_i)$ as follows. Let S be the set of sets of words currently in $WD(B_i)$. When a new set of words s arrives, BestFitTrie selects the best trie in the forest of tries of $WD(B_i)$, and the best location in that trie to insert s. The algorithm for choosing t depends on the current organization of the word directory and will be given below.

Throughout its existence, each trie T of $WD(B_i)$ has the following properties. The nodes of T store sets of words and other data items related to these sets. Let $sets\text{-}of\text{-}words(T)$ denote the set of all sets of words stored by the nodes of T. A node of T stores more than one set of words if and only if these sets are identical. The root of T (at depth 0) stores sets of words with an identifying subset of cardinality

one. In general, a node n of T at depth i stores sets of words with an identifying subset of cardinality $i+1$. A node n of T at depth i storing sets of words equal to s is implemented as a structure consisting of the following fields:

- $Word(n)$: the $(i+1)$-th word w_i of identifying subset $\{w_0, \ldots, w_{i-1}, w_i\}$ of s where $w_0, \ldots, w_{i-1}$ are the words of nodes appearing earlier on the path from the root to node n.

- $Query(n)$: a linked list containing the identifier of query q that contained word pattern wp for which $\{w_0, \ldots, w_i\}$ is the identifying subset of $sets\text{-}of\text{-}words(T)$.

- $Remainder(n)$: if node n is a leaf, this field is a linked list containing the words of s that are not included in $\{w_0, \ldots, w_i\}$. If n is not a leaf, this field is empty.

- $Children(n)$: a linked list of pairs (w_{i+1}, ptr), where w_{i+1} is a word such that $\{w_0, \ldots, w_i, w_{i+1}\}$ is an identifying subset for the sets of words stored at a child of w_i and ptr is a pointer to the node containing the word w_{i+1}.

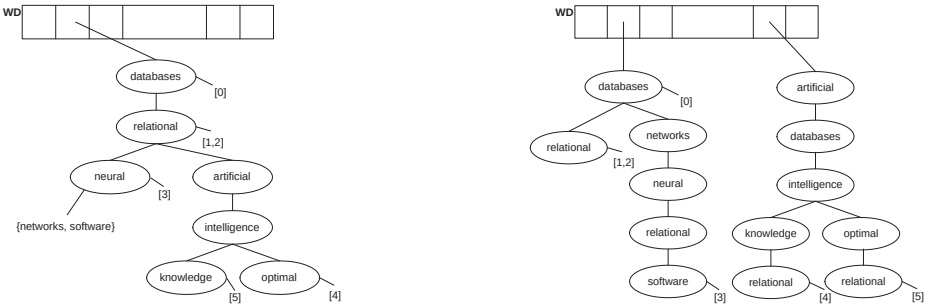
The sets of words stored at node n of T are equal to $\{w_0, \ldots, w_n\} \cup Remainder(n)$, where $w_0, \ldots, w_n$ are the words on the path from the root of T to n. An identifying subset of these sets of words is $\{w_0, \ldots, w_n\}$. Figure 6.1(a) shows the general form of our index structure (we have omitted ET and PA). The part of $WD(B_i)$ corresponding to the queries of Table 6.1 is shown in full including lists $Query(n)$ and $Remainder(n)$. The purpose of $Remainder(n)$ is to allow for the delayed creation of nodes in trie. This delayed creation lets us choose which word from $Remainder(n)$ will become the child of current node n depending on the sets of words that will arrive later on.

The algorithm for inserting a new set of words s in a word directory is as follows. The first set of words to arrive will create a trie with a randomly chosen word as the root and the rest stored as the remainder. The second set of words will consider being stored at the existing trie or create a trie of its own. In general, to insert a new set of words s, BestFitTrie iterates through the words in s and utilises the hash table implementation of the word directory to find all *candidate tries* for storing s: the

tries with root a word of s. To store sets as compactly as possible, BestFitTrie then looks for a trie node n such that the set of words $(\{w_0, \ldots, w_n\} \cup Remainder(n)) \cap s$, where $\{w_0, \ldots, w_n\}$ is the set of words on the path from the root to n, has maximum cardinality. There may be more than one node that satisfies this requirement and such nodes might belong to different tries. Thus BestFitTrie performs a depth-first search down to depth $|s| - 1$ in *all* candidate tries in order to decide on the optimal node n. The path from the root to n is then extended with new nodes containing the words in $\tau = (s \backslash \{w_0, \ldots, w_n\}) \cap Remainder(n)$. If $s \subseteq \{w_0, \ldots, w_n\} \cup Remainder(n)$, then the last of these nodes l becomes a new leaf in the trie with $Query(l) = Query(n) \cup \{q\}$ (q is the new query from which s was extracted) and $Remainder(l) = Remainder(n) \backslash \tau$. Otherwise, the last of these nodes l points to two child nodes l_1 and l_2. Node l_1 will have $Word(l_1) = u$, where $u \in Remainder(n) \setminus \tau$, $Query(l_1) = Query(n)$ and $Remainder(l_1) = Remainder(n) \backslash (\tau \cup \{u\})$. Similarly node l_2 will have $Word(l_2) = v$, where $v \in s \setminus (\{w_0, \ldots, w_n\} \cup \tau)$, $Query(l_2) = q$ and $Remainder(l_2) = s \setminus (\{w_0, \ldots, w_n\} \cup \tau \cup \{u\})$. The complexity of inserting a set of words in a word directory is *linear* in the size of the word directory.

The filtering procedure utilises two arrays named $Total$ and $Count$. $Total$ has one element for each query in the database and stores the number of atomic formulas contained in that query. Array $Count$ is used for counting how many of the atomic formulas of a query match the corresponding attributes of a document. Each element of array $Count$ is set to zero at the beginning of the filtering algorithm. If at algorithm termination, a query's entry in array $Total$ equals its entry in $Count$, then the query matches the published document, since all of its atomic formulas match the corresponding document attributes.

When a document d is published at the server, filtering proceeds as follows. BestFitTrie hashes the text value $C(d)$ contained in each document attribute C and probes the ET to find matching atomic formulas with equality. Then for each attribute C in $DAL(d)$ and for each word w in $DWL(C)$, the trie of $WD(C)$ with root w is traversed in a breadth-first manner. Only subtrees having as root a word contained in $C(d)$ are examined, and hash table $OT(d)$ is used to identify them quickly. At each node n of the trie, the list $Query(n)$ gives implicitly all atomic



(a) BestFitTrie (b) PrefixTrie

Figure 6.1: BestFitTrie vs. PrefixTrie for the atomic queries of Table 6.1

formulas $C \sqsupseteq wp_i$ that can potentially match $C(d)$ if the proximity formulas in wp_i are also satisfied. This is repeated for all the words in $DWL(C)$, to identify all the qualifying atomic formulas for attribute C. Then the proximity formulas for each qualifying query are examined using a recursive polynomial time algorithm that takes a proximity formula and a text value as an input, and decides whether the proximity formula is satisfied by the text value. For each atomic formula satisfied by $C(d)$, the corresponding query element in array *Count* is increased by one. At the end of the filtering algorithm arrays *Total* and *Count* are traversed and the values stored for each query are compared. The equal entries in the two arrays give us the queries satisfied by d.

6.1.2 Other Filtering Algorithms

To evaluate the performance of BestFitTrie, we have also implemented algorithms BF, SWIN and PrefixTrie. BF (Brute Force) has no indexing strategy and scans the query database sequentially to determine matching queries. SWIN (Single Word INdex) utilises a two-level index for accessing queries in an efficient way. A query of the form presented at the beginning of this section is indexed by SWIN under all its attributes $A_1, \ldots, A_n, B_1, \ldots, B_m$ and also under n text values $s_1, \ldots, s_n$ and m words selected randomly from $wp_1, \ldots, wp_m$. More specifically SWIN utilises an ET to index equalities and an AD pointing to several WDs to index the atomic containment queries. Atomic queries within a WD slot are stored in a list. PrefixTrie is an extension of the algorithm Tree of [209] appropriately modified to cope with attributes and proximity information. Tree was originally proposed for storing *con-*

junctions of keywords in secondary storage in the context of the SDI system SIFT. Following Tree, PrefixTrie uses *sequences* of words sorted in lexicographic order for capturing the words appearing in the word patterns of atomic formulas (instead of sets used by BestFitTrie). A trie is then used to store sequences compactly by exploiting *common prefixes* [209].

Algorithm BestFitTrie constitutes an improvement over PrefixTrie. Because PrefixTrie examines only the prefixes of sequences of words in lexicographic order to identify common parts, it misses many opportunities for clustering (see Figure 6.1). BestFitTrie keeps the main idea behind PrefixTrie but (a) handles the words contained in a query as a set rather than as a sorted sequence and (b) searches exhaustively the forest of trie to discover the best place to introduce a new set of words. This allows BestFitTrie to achieve better clustering as shown in Figures 6.1(a) and 6.1(b), where we can see that it needs only one trie to store the set of words for the formulas of Table 6.1, whereas PrefixTrie introduces redundant nodes that are the result of using a lexicographic order to identify common parts. This node redundancy can be the cause of deceleration of the filtering process as we will show in Section 6.4.

6.2 Reorganisation of Queries

Most of the algorithms presented earlier use heuristics to identify and cluster similar queries, in order to achieve better performance during matching. These heuristics provide an organisation of queries that is dependent on the order of *insertion* of the queries in the system. Taking BestFitTrie as an example, we can notice that for a given set of user queries Q, and two different orderings of these queries, the resulting clustering that is achieved is different. In other words, if we consider the clustering problem as a search problem over the search space of all possible query organisations, then BestFitTrie actually provides us with a greedy heuristic that results in a possibly non-optimal solution (but yet a fairly good one as we will see in Section 6.4). An alternative to organising the user queries in a heuristic fashion is to search over the space of all possible organisations for the optimal one. This is obviously prohibitively expensive and will not be considered further in this work.

In this section we propose ReTrie, an algorithm that can be used together with BestFitTrie in order to reorganise the query database periodically and achieve better clustering. To describe ReTrie, we will need the following definitions.

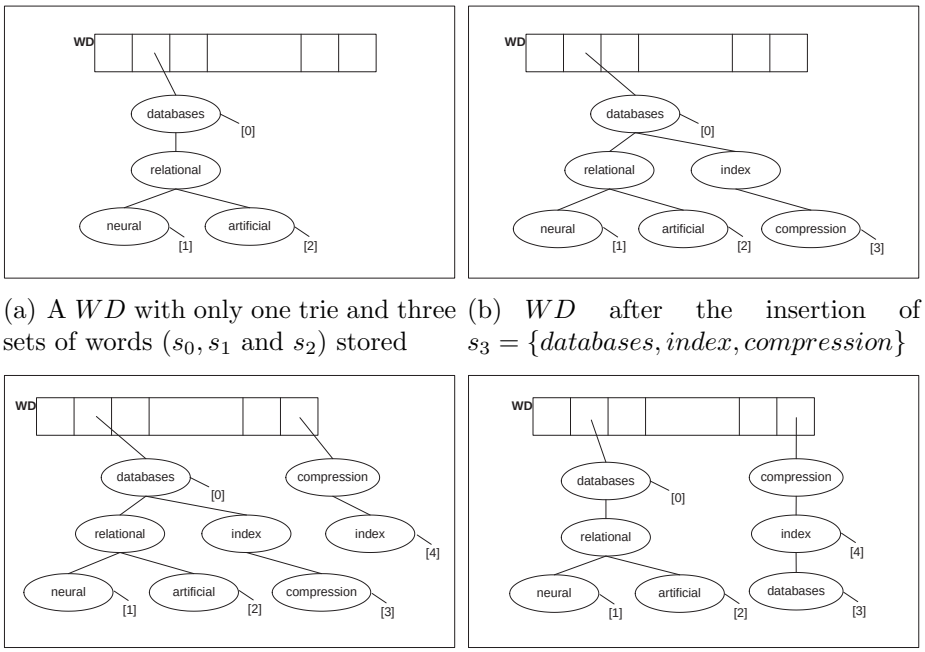
Definition 12 *Let s be a set of words indexed at trie node n of trie T. For this set of words, we have that $s = \{w_0, \ldots, w_n\} \cup Remainder(n)$, where $w_0, \ldots, w_n$ are the words in the path from the root of T to node n. The* clustering ratio *of s, denoted as $ClusteRat(s)$, is $ClusteRat(s) = \frac{|\{w_0,\ldots,w_n\}|}{|s|}$.*

$ClusteRat(s)$ is used to quantify the notion of the clustering quality for a given set of words. From the definition, it follows that $0 < ClusteRat(s) \leq 1$. Generally when $ClusteRat(s)$ is near 0, the set of words s is considered badly clustered and ReTrie should try repositioning it in the forest of tries. On the contrary, when $ClusteRat(s)$ is near 1, the set of words is considered highly clustered and no repositioning is needed. Below we define when a set of words is considered under-clustered.

Definition 13 *Let s be a set of words indexed at trie node n of trie T. s is considered* under-clustered *iff $ClusteRat(s) < c$, where $0 \leq c \leq 1$ is a* clustering threshold.

To keep track of the clustering ratio of each set of words s, ReTrie utilises a *clustering array* (CA) that contains an entry for every set of words inserted in $WD(B_i)$. Each CA entry contains a pointer to the position s is currently stored and a number representing $ClusteRat(s)$. When a new set of words s is indexed at node n of trie T, the clustering ratio of s in CA is initialised using the formula from Definition 12. Additionally, if $Remainder(n)$ is expanded to create new nodes, then the clustering ratios of the other sets of words stored at n should be updated, since they now have more of their words clustered. If $Remainder(n)$ is not expanded, no other update is necessary to array CA.

Algorithm ReTrie, presented in Figure 6.3, is run periodically in order to reposition badly clustered sets of words. All under-clustered sets of words (identified by scanning CA) are candidates for moving when the algorithm is executed. For each under-clustered set of words s with clustering ratio $ClusterRat(s)$, ReTrie searches



(a) A WD with only one trie and three sets of words (s_0, s_1 and s_2) stored

(b) WD after the insertion of $s_3 = \{databases, index, compression\}$

(c) WD after the insertion of $s_4 = \{compression, index\}$

(d) WD after ReTrie has re-organised the index

Figure 6.2: Profile insertions and re-organisation achieved by ReTrie

the forest of tries of $WD(B_i)$ looking for all nodes that can store s. For each one of these positions, ReTrie calculates the new clustering ratio for s and chooses the position that results in the maximum clustering ratio, $maxClu(s)$. s is moved to the new position found only if $ClusterRat(s) < maxClu(s)$. Moving s to a new position results in updates to array CA. The update procedure is the same as the one described in the previous paragraph for query insertions.

ReTrie can improve the clustering of queries because not all queries have the same clustering opportunities when entering the query index with BestFitTrie. This can be explained as follows. When a new set of words s needs to be indexed, the clustering algorithm looks for a trie in the forest of tries of $WD(B_i)$ and a node in that trie, such that $ClusterRat(s)$ is maximised. It is easy to see that the number of tries and the number of nodes in those tries affect the clustering opportunities of s: the higher the number of candidate positions to insert s, the higher the possibility for the algorithm to cluster s effectively (that is in a position with a higher $ClusterRat(s)$). This is shown by the example of Figure 6.2. Consider a forest consisting of a single trie currently indexing three sets of words: $s_0 = \{databases\}$, $s_1 = \{databases, relational, neural\}$ and $s_3 = \{databases, relational, artificial\}$ (Figure

Algorithm *ReTrie*

```
01: begin
02:   for i = 0 to N do                                  ▷ for all stored sets of words
                                                           (N: # of stored sets of words)
03:     if CA[i].clusterRat < c then                     ▷ identify under-clustered ones
04:       let s be the i-th set of words
05:       for each trie T such that root(T) ∈ s do       ▷ for all candidate tries
06:         for each node n ∈ T such that word(n) ∈ s do ▷ for all possible storage
                                                           positions in candidate tries
07:           calculate clusterRat(s)
08:           if clusterRat(s) > curClusterRat then      ▷ if a better position is
                                                           found make a note of it
09:             curClusterRat ← clusterRat(s)
10:             curPosition ← n
11:           end if
12:         end for
13:       end for
14:     end if
15:     if curPosition ≠ CA[i].position then             ▷ if the best position found is
                                                            better than the initial
16:       move s to curPosition                          ▷ move s there
17:       CA[i].position ← curPosition                   ▷ and update CA
18:       CA[i].clusterRat ← curClusterRat
19:     endif
20:   end for
21: end
```

Figure 6.3: Pseudocode for algorithm ReTrie

6.2(a)). When a set of words $s_3 = \{databases, index, compression\}$ arrives, it is inserted in the only position available and is clustered only under one word (Figure 6.2(b)). Now $ClusterRat(s_3) = 0.33$. Upon arrival of $s_4 = \{index, compression\}$ (Figure 6.2(c)), it is obvious that there is a better position to index s_3. This position is together with s_4, as it is shown in Figure 6.2(d) where $ClusterRat(s_3) = 0.66$. Notice that in the forest of tries of $WD(B_i)$ shown in Figure 6.2(c), words *index* and *compression* appear in two nodes each (this *redundancy* in nodes is one of the factors that slow down filtering), whereas after the re-organisation of the forest (Figure 6.2(d)) there are no redundant nodes for these words (i.e., they appear only once in the forest). Generally, it is not possible to remove all redundant nodes in a forest (notice for example the word *database* in this example), so our effort concentrates on minimising these nodes by re-organisation.

There are two major problems when trying to design an algorithm for reorganising the database. The first problem is what to reorganise and the second is when you should do this reorganisation. We have already presented our solution to the first problem, and now we discuss the second one.

The straightforward approach is to reorganise the database after every fixed number of insertions or at fixed time intervals. This approach has the drawback of ignoring the clustering criteria, and invokes reorganisation even at cases where there is not much need for it. Another option we considered is defining a criterion for the quality of clustering and reorganise when the quality of clustering drops. We have tried two different approaches for defining this criterion. The first approach considers the quality of clustering in correlation with the number of redundant nodes in the forest. This approach exploits the fact that badly clustered queries introduce more redundant nodes (to see this consider the number of nodes in Figures 6.2(c) and (d)). The second approach quantifies the clustering quality in terms of the percentage of under-clustered queries in the system. In both approaches, an appropriate threshold is used to trigger the reorganisation process. Finally, a simple, effective and widely used (in other related cases) approach is to consider reorganisation when the system is idle. This approach, apart from the obvious benefits of not overloading the system, offers the advantage that the reconstruction of the database can be done partially, which is useful if we want to exploit even small time intervals to do the heavy work of reorganisation.

Exploring all the above possibilities is out of the scope of this work and will not be considered further. For the experiments presented in the next section, we used the percentage of under-clustered queries to trigger the reorganisation process.

6.3 Filtering Algorithms for $\mathcal{AWPS}$

To index $\mathcal{AWPS}$ queries we utilise a combination of two different data algorithms: our home-brewed BestFitTrie algorithm, presented in Section 6.1.1 and algorithm SQI presented in detail in [211]. In this section we briefly present the SQI algorithm that is used for indexing VSM queries and discuss how the two different algorithms can be used together to solve the filtering problem for queries in $\mathcal{AWPS}$.

The idea behind SQI is to use an inverted index to store the most significant word contained in an atomic formula's text value. Thus, for a VSM query of the form $A_1 \sim_{a_1} s_1 \wedge ... \wedge A_n \sim_{a_n} s_n$, only some carefully selected terms contained in text values $s_1, ..., s_n$ will be chosen and the query is index under those terms only. Notice that usually these terms are the most discriminative terms contained in the text values, which means that they are typically terms that appear less frequently in documents. This way indexing a query under all its terms is avoided and faster times can be achieved during filtering. Notice also that this indexing is done without sacrificing the accuracy when computing the relevance of an incoming document to a query.

The above method together with BestFitTrie is used at each node to index locally the queries it is responsible for. Two arrays similar to arrays $Total$ and $Count$ are used to keep track of the number of atomic formulas contained in a query and the number of the atomic formulas of a query that matched the corresponding attributes of an incoming document. If at algorithm termination, a query's entry in array $Total$ equals its entry in $Count$, then the query matches the published document, since all of its atomic formulas match the corresponding document attributes. Note also that each incoming document has to be matched against both algorithms (BestFitTrie and SQI), to decide which queries it satisfies.

We have not conducted and experiments regarding the performance of algorithm SQI, but the interested reader is referred to [211]. It remains open whether the two algorithms can be combined in a single trie-like data structure that will allow faster filtering of incoming documents.

6.4 Experimental Evaluation

To carry out the experimental evaluation of the algorithms described in the previous section, we needed data to be used as incoming documents, as well as user queries. It may not be difficult to collect data to use in the evaluation of filtering algorithms for SDI scenarios. For the model $\mathcal{AWP}$ considered in this work, there are various document sources that one could consider: meta-data for papers on various publisher Web sites (e.g., ACM or IEEE), electronic newspaper articles, articles

Description	Value
Number of documents	10,426
Document vocabulary size	641,242
Maximum document size (words)	110,452
Minimum word size (letters)	1
Maximum word size (letters)	35

Table 6.2: Some characteristics of the NN corpus

Attribute	Percentage of documents
TITLE	63%
AUTHORS	58%
ABSTRACT	88%
BODY	86%
YEAR	63%

(a)

# of attributes	Percentage of documents
1	7.9%
2	28.7%
3	2.4%
4	16.0%
5	45.0%

(b)

Table 6.3: Some attribute characteristics of the corpus documents

from news alerts on the Web (e.g., `http://www.cnn.com/EMAIL`) etc. However, it is rather difficult to find user queries except by obtaining proprietary data (e.g., from CNN's news alert system).

For our experiments we chose to use the NN corpus, also used for the experiments in the previous chapter. Table 6.2 summarises some key characteristics of the document corpus, where Tables 6.3(a) and 6.3(b) give details on the fraction of documents that contain each attribute, and on the fraction of documents that contain a specific number of attributes respectively. Because no database of queries was available to us, we developed a methodology for creating user queries using *words* and *technical terms* extracted automatically from the Research Index documents using the C-value/NC-value approach of [82]. The extracted multi-word technical terms are used to create proximity formulas and also as conjunctions of keywords in user queries. For the formulation of user queries author names and words extracted from paper abstracts are also used. The attribute universe for the experiments presented in this section consists of attributes *paper title, authors, abstract* and *body*.

The basic concept for the query creation in our methodology is that of a *unit*. Atomic queries are created as conjunctions of units selected uniformly from unit sets, whereas queries are created as conjunctions of atomic queries selected from the attribute universe with a probability p_{C_i}. In our scenario four different types of unit sets exist:

- *Author unit set.* This set contains the last names of authors appearing in the full citation graph of ResearchIndex[1]. Each author appears in the author unit set as many times as the in-degree of the papers he has published. Thus the probability $P(\alpha)$ of author α to appear in a query is $P(\alpha) = N_\alpha / \sum_{k \in V_\alpha} N_k$, where N_α is the number of papers in the citation graph that cite the work of author α, and V_α is the author vocabulary obtained also by the full citation graph.

- *Proximity formulas unit set.* This set contains proximity formulas created using the extracted multi-word terms. The technical terms with more than five words were excluded since they were noise, and the set was produced after applying upper and lower NC-value cut-off thresholds for the remaining terms. The proximity operators in this set contain distances according to the number of words contained in each multi-word term.

- *Keywords from technical terms.* This unit set contains keywords extracted from technical terms. These keywords are used as conjuncts in the creation of atomic queries.

- *Nouns from abstracts.* This set contains the nouns used in the corpus abstracts after the cut-off of the most and least frequent words. The rationale behind this is that abstracts are intended to be a comprehensive summary of the publication content, thus nouns from abstracts are appropriate candidates for use in queries.

An example of a user query created synthetically from the methodology briefly sketched above is:

[1]The citation graph contains information about the citations between research papers and was compiled in [143].

Parameter	Description
W	average # of words per document
W_d	average # of distinct words per document
N	# of queries in database
D	# of incoming documents
m	percentage of stored queries matching the incoming documents
c	clustering threshold

Table 6.4: Parameters varied in experiments and their descriptions

$$(\text{AUTHOR} \sqsupseteq \mathit{Riedel}) \quad \wedge$$
$$(\text{TITLE} \sqsupseteq \mathit{implementation} \wedge (\mathit{RBF} \prec_{[0,3]} \mathit{networks}))$$

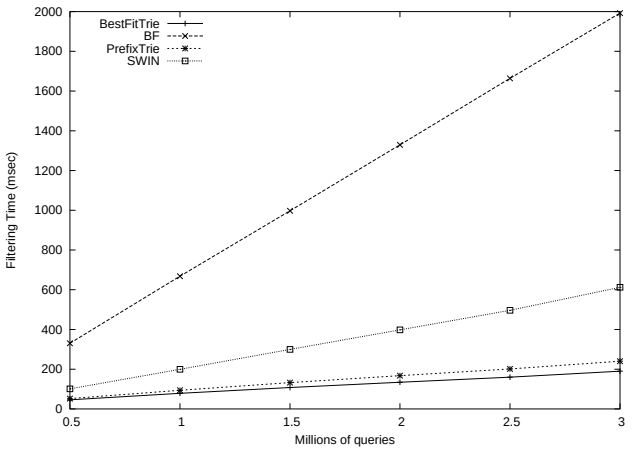
For more details of the methodology the interested reader can refer to [198]. As a byproduct of our work on this topic, we have a new corpus of documents and continuous queries which is representative of digital library scenarios, and can also be used by other researchers in the area of information filtering.

All the algorithms were implemented in C/C++, and the experiments were run on a PC, with a Pentium III 1.7GHz processor, with 1GB RAM, running Linux. Time shown in the graphs is wall-clock time and the results of each experiment are averaged over 10 runs to eliminate any fluctuations in the time measurements.

6.4.1 Varying the Database Size

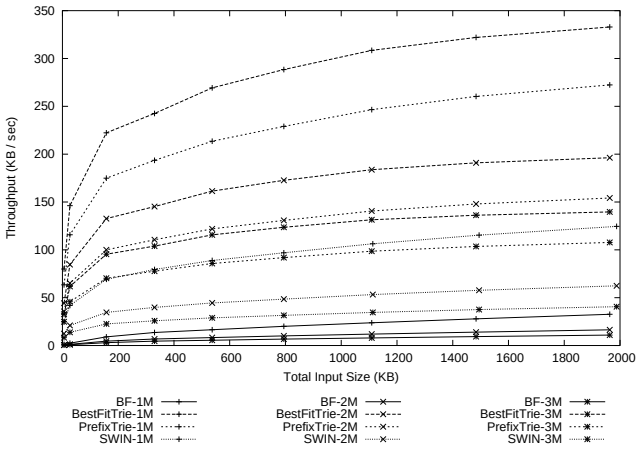
The first experiment that we conducted to evaluate our algorithms targeted the performance of the four algorithms under different sizes of the query database. In this experiment we randomly selected one hundred documents from the NN corpus and used them as incoming documents in the query databases of different sizes. The size and the matching percentage for each document used was different but the average document size was 6869 words, whereas on average 1% of the queries stored matched the incoming documents.

As we can see in Figure 6.4, the time taken by each algorithm grows linearly with the size of the query database. However SWIN, PrefixTrie and BestFitTrie are less sensitive than Brute Force to changes in the query database size. The trie-



Parameter	Value
W	6869
W_d	1115
N	0.5M-3M
D	100
m	1%
c	-

Figure 6.4: Effect of the query database size in filtering time

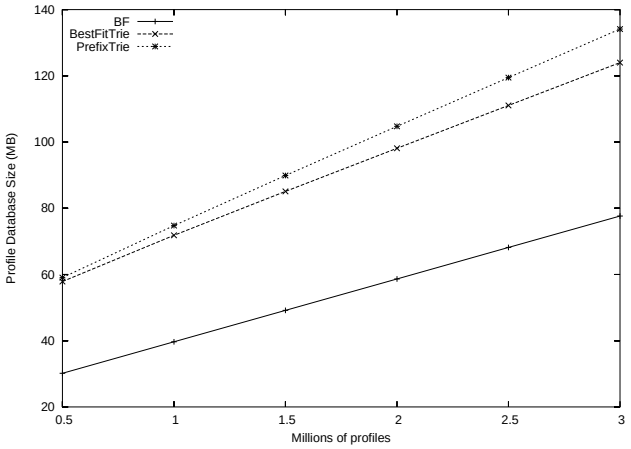


Parameter	Value
W	6869
W_d	1115
N	1M-3M
D	100
m	1%
c	-

Figure 6.5: Performance in terms of throughput for the algorithms of Section 6.1

based algorithms outperform SWIN mainly due to the clustering technique that allows the exclusion of more non-matching atomic queries filtering. We can also observe that the better exploitation of the commonalities between queries improves the performance of BestFitTrie over PrefixTrie, resulting in a significant speedup in filtering time for *large query databases*. Additionally, Figure 6.5 contrasts the algorithms in terms of throughput were we can see that BestFitTrie gives the best filtering performance managing to process a load of about 150KB per second for a query database of 3 million queries.

In terms of space requirements, BF needs about 50% less space than the trie-based algorithms, due to the simple data structure that poses small space requirements. Additionally the rate of increase for the two trie-based algorithms is similar to that of BF, requiring a fixed amount of extra space each time. Figure 6.6 shows



Parameter	Value
W	-
W_d	-
N	0.5M-3M
D	-
m	-
c	-

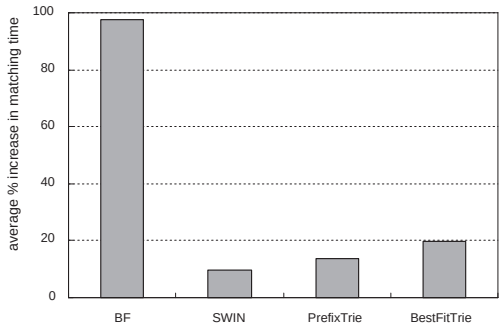
Figure 6.6: Space requirements for the trie-based algorithms

the space requirements of the trie-based algorithms in comparison to the brute force approach. From the experiments above, it is clear that BestFitTrie speeds up the filtering process with a small extra storage cost, and proves faster than the rest of the algorithms, managing to filter as much as 3 million queries in less the 200 milliseconds, which is about 1000% times faster than the sequential scan method and 20% faster than PrefixTrie.

6.4.2 Varying the Matching Percentage

In this experiment we wanted to observe the sensitivity of each algorithm when the number of queries that matched incoming documents varies. To do this, we used two sets of documents, A and B, that contained about the same number of distinct words and the same attributes, but the number of queries that matched each document was different. Notice that given the way the algorithms are designed, the important parameter of a document is the number of distinct words contained, rather than its size. This happens because the probing of the query index uses the distinct words contained in the attribute text. Practically, the increase in the number of distinct words increases the probability of a specific word contained in a query, to be also contained in the incoming document. This in turn increases the number of queries with proximity formulas that need to be evaluated[2], which is a time consuming

[2]Remember that the evaluation of an atomic query is done in two phases; the existence of keywords is checked first and the evaluation of the proximity formulas follows.



Parameter	Value for set A	Value for set B
W	47155	42419
W_d	2920	3033
N	3M	3M
D	10	10
m	2%	22%
c	-	-

Figure 6.7: Average % increase in filtering time for a 20% increase in the number of matching queries

process. The size of the document is of smaller importance, since it only increases the number of positions of a specific word in the document, and thus the number of checks at proximity evaluation time. However due to the algorithm presented in [123], the majority of the positions of a specific word in a document can be excluded from the proximity evaluation.

Figure 6.7 shows the % increase in matching time for two documents A and B with the same number of distinct words, but different number of queries matching them. Document set B contained 47155 words on average, and matched 20% more queries than document set A, which contained 42419 words on average. Both document sets contained four $(attribute, value)$ pairs, and the query database contained 3 million queries. Apart from BF which showed a 97% increase in the matching time, BestFitTrie appears to be the most sensitive to the increase in the matching percentage (showing a 19% increase in filtering time), in contrast to PrefixTrie and SWIN, which appear to be less affected (with 13% and 9% increase respectively). This can be explained as follows. The trie structure of PrefixTrie and BestFitTrie forces them to explore a big number of child nodes when a word node appears in a document, in contrast to SWIN that searches in either case all the nodes that are hashed under a specific word. This means that in higher matching percentages, the trie-based algorithms loose some of the advantages offered by their sophisticated data structures and show greater sensitivity to the matching degree. However the trie-based algorithms are still significantly faster, with BestFitTrie being the faster algorithm of all four despite the high increase.

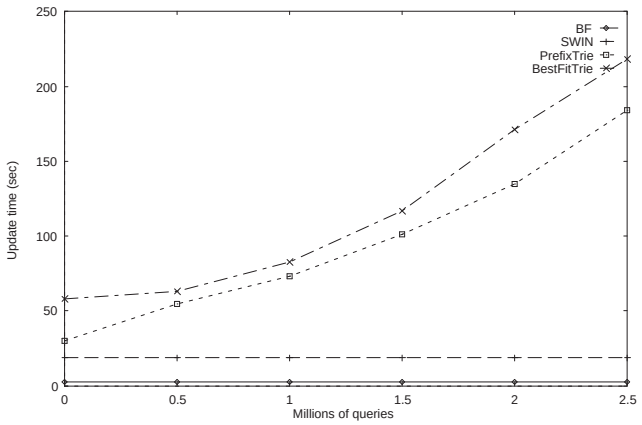
6.4.3 Varying the Document Size

In this experiment we want to observe the behaviour of the four algorithms when the size of the incoming document increases. This time two sets of documents with about the same number of queries matching them were chosen, and the variations in the performance of the four algorithms were examined. Document set A contained 10 documents of average length of 75344 words and 1864 distinct words, whereas document set B contained 10 documents of average length of 148609 words and 2304 distinct words, that is about 24% more distinct words than document set A. The differences in the performance were below 5% in matching time for SWIN, PrefixTrie and BestFitTrie, whereas BF showed an increase of about 50%. The insensitivity of SWIN, PrefixTrie and BestFitTrie in the document size is mainly due to the hash representation of the document and the way the matching process is carried out. During the matching process, we actually consider only the distinct words of the document (that are obviously significantly less than the document itself for large documents), and check the existence of a word in the document using a hash function, which provides fast answer times. Moreover the proximity evaluation is not greatly affected from the large number of word positions inside a document due to the well-designed proximity evaluation algorithm of [123] that allows the omission of a large number of word positions in a document.

Since in an SDI scenario one may not always have to deal with large documents (for example, if $\mathcal{AWP}$ is used for describing metadata about research papers) we carried out experiments with documents with smaller size. More specifically experiments with documents of mean size of 551 words, show that BestFitTrie performs even better in terms of filtering time, being 1.75 times faster that PrefixTrie and about 86 times faster that BF (as opposed to 1.2 and about 10 times faster respectively for documents of mean size 6869 words).

6.4.4 Updating the Query Database

In this experiment we investigated the update time for the four different algorithms. To measure the average time needed to insert a single query in the database we



Parameter	Value
W	-
W_d	-
N	0M-2.5M
D	-
m	-
c	-

Figure 6.8: Query insertion time for different query database sizes

worked as follows. Starting with the empty database, we measured the total time needed to populate it with 500K queries, and proceeded in a similar way by adding batches of 500K queries in our database and measuring the total insertion time per batch. Subsequently the average insertion time per query for a given batch of queries can be found simply by dividing the total time measured with the population of the batch to produce a single point in the graph of Figure 6.8.

It should be clear that for BF and SWIN the query insertion time will be constant on average, since BF does a simple insertion at the end of a list, while SWIN utilises a hash table and inserts each atomic query in the beginning of the list pointed to by the hash table slot. For the trie-based algorithms the query insertion is a more complex process that involves the examination of lists at every level of the trie. While PrefixTrie examines only a single path in a single trie of the forest, BestFitTrie needs to examine several paths in the trie and also several tries (in the usual case as many tries as the number of words in a profile). Our remarks are verified by the graph in Figure 6.8 that shows the average insertion time in milliseconds for a query q for a given database size. In this figure we can see that BestFitTrie needs about 20% more time than PrefixTrie to insert a query in a database with 2.5 million profiles. This is a standard tradeoff where the algorithm spends some extra time at indexing to save it at query execution.

6.4.5 Incorporating Ranking Information

To examine the performance of the two trie-based algorithms (namely PrefixTrie and BestFitTrie), we modified them in order to take into account information about the frequency of occurrence of words in documents. More specifically we use the *document frequency* of a word w_i (denoted by df_i), which represents the number of documents in a collection that contain w_i, to identify the frequent and infrequent words among the documents. In an SDI scenario where no document collection is available, we can compute df_i on the collection of recently processed documents [208] (say k most recent documents arrived, where k is large enough). Using this information, we created variations of the trie-based algorithms that use different heuristics for storing user queries in tries.

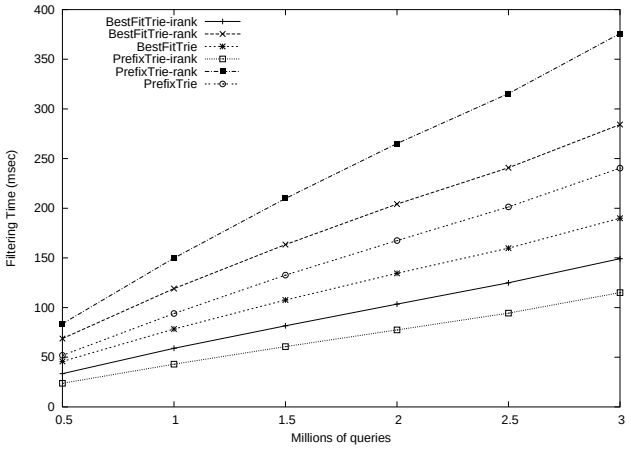
The *rank* heuristic stores the most frequent words among the documents (that is the words with the highest *df*) near the roots of the tries, while the less frequent words (that is the words with the lowest *df*) are pushed deeper in them resulting in relatively few big and "wide" tries (since their roots will exist in more queries). The algorithms using the rank heuristic are PrefixTrie-rank and BestFitTrie-rank.

Contrary to *rank*, the *inverse rank* heuristic (*irank*) [208] stores the least frequent words of the queries near the roots of the tries, while the frequent ones are pushed deeper in the tries, resulting in many narrow tries. Thus more queries are put in subtrees of words occurring less frequently, resulting in less lookups during filtering time. The algorithms using the irank heuristic are PrefixTrie-irank and BestFitTrie-irank.

The probability that any word w_i appears in an incoming document d is defined to follow probability distribution $D(w_i)$, where $0 \leq D(w_i) \leq 1$. The number of nodes N that will be examined within each trie depends on the clustering heuristic and is equal to

$$N = D(w_1)N_1 + D(w_2)N_2 + \ldots + D(w_{|V|})N_{|V|} \tag{6.1}$$

where N_i is the number of nodes in the trie that have word w_i as root node, and $|V|$



Parameter	Value
W	6869
W_d	1115
N	0.5M-3M
D	100
m	1%
c	-

Figure 6.9: Incorporating word frequency information into the trie-based algorithms, and its effect in filtering time

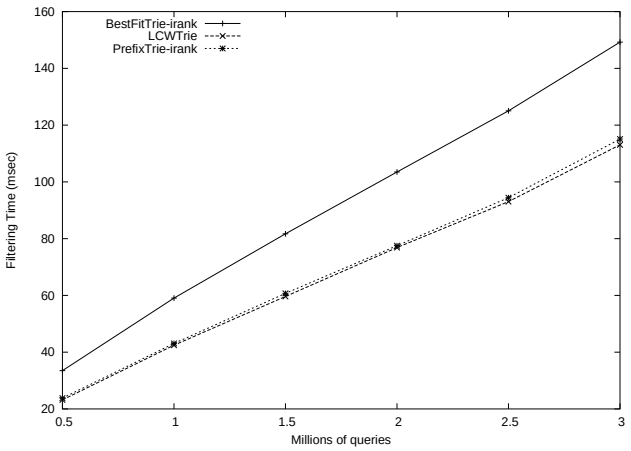
is the size of our vocabulary. The sum

$$N_1 + N_2 + \ldots + N_{|V|} \tag{6.2}$$

is positive and it is always less than or equal to the number of words in the query database.

From Equations 6.1 and 6.2 we can see that the number N of nodes examined is minimised if we assign more words to WD slots pointing to words (trie roots) with smaller probability to appear in a document. Based on the above observation, we created a modification of BestFitTrie, called LCWTrie (Least Common Word Trie,) by limiting BestFitTrie to consider only one candidate trie during insertion: the one that has the least frequent word of the atomic query as root. In this way, the atomic query can only be inserted in that trie (or that trie will be created if it does not exist), while the remainder of the words of the atomic query will be organised following the insertion algorithm of BestFitTrie (this will give us the best organisation considering only this trie instead of the whole forest).

In Figure 6.9 we present the performance of PrefixTrie and BestFitTrie and their ranking variations. We can see that using the rank heuristic the performance of both algorithms deteriorates, due to the creation of large tries that need bigger exploration time. We can also observe that the irank heuristic improves the performance of both trie-based algorithms, with the greater effect shown on PrefixTrie that becomes



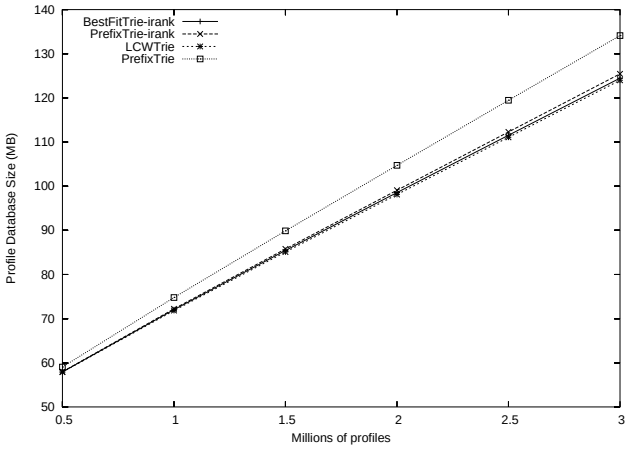
Parameter	Value
W	6869
W_d	1115
N	0.5M-3M
D	100
m	1%
c	-

Figure 6.10: Performance of LCWTrie in comparison to the two faster filtering algorithms

faster than BestFitTrie-irank. This improvement in performance for both algorithms was expected as shown earlier in this section.

Figure 6.10 presents the performance of the three faster algorithms, namely PrefixTrie-irank, BestFitTrie-irank and LCWTrie. BestFitTrie-irank prioritises clustering over frequency information by examining all candidate tries and choosing the one that has the most common words. Word frequency information plays a secondary role, allowing the algorithm to choose between tries with the same common words the trie that has the highest ranked word as a root. On the other hand, PrefixTrie-irank and LCWTrie are designed to show a preference in frequency information against clustering. Both algorithms examine exactly one candidate trie, that with the least frequent word as root. Additionally, LCWTrie organises the query within that trie in the best possible way, taking into account common words between the queries already stored. In contrast, PrefixTrie-irank does not care about clustering and stores the query according to frequency information only, that is the word with the lowest rank goes deeper in the trie.

Our observations about the significance of frequency information presented in the beginning of the section are verified. From the experiments of Figure 6.10, we see that LCWTrie performs similarly with PrefixTrie-irank, although it presents a slight advantage for large query databases, due to the clustering within the trie. Additionally, both algorithms outperform BestFitTrie that owes its performance mainly to clustering, giving little consideration to frequency information. Figure



Parameter	Value
W	-
W_d	-
N	0.5M-3M
D	-
m	-
c	-

Figure 6.11: Memory requirements of ranking variations of BestFitTrie and PrefixTrie

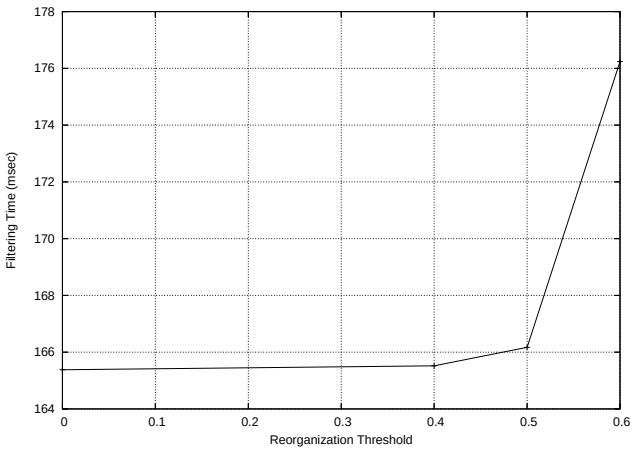
6.11 shows the memory requirements of the different algorithms for varying database sizes. The improvement in clustering using the heuristics discussed here is obvious. All algorithms that exploit frequency information need significantly less space to store the same number of queries compared to PrefixTrie. This difference in storage requirements comes from the existence of less redundant nodes.

6.4.6 Reorganisation of Queries

To study the effect of our reorganisation strategy to the performance of the algorithms at filtering time, we compared ReTrie with BestFitTrie. The ranking counterparts of BestFitTrie presented earlier (namely BestFitTrie-rank and -irank) are not suitable for comparing with our reorganisation strategy, because of the non-flexible way of clustering they use. The rank and irank heuristic do not allow for many alternatives to cluster a set of words, resulting in the inapplicability of the ReTrie algorithm. For the same reasons, ReTrie cannot be used in conjunction with PrefixTrie.

Varying the Clustering Threshold

In this experiment we wanted to determine a baseline value for the clustering threshold c to be used in the evaluation of ReTrie. To do so, we populated the query database *db* with 2.5 million queries and invoked ReTrie with different reorganisation thresholds ranging from 0 to 0.6 with an increase step of 0.1. For each reorgan-



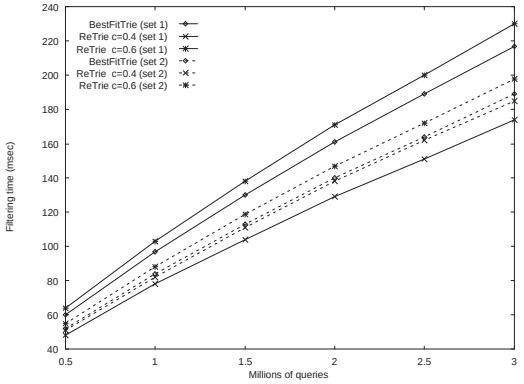
Parameter	Value
W	6869
W_d	1115
N	2.5M
D	100
m	1%
c	0-0.6

Figure 6.12: Filtering time for different clustering thresholds

isation threshold, a different re-organised database db' was created, since different sets of words were chosen to be moved. Then, we randomly selected one hundred documents from the NN corpus as incoming documents to be matched against each one of the different databases. The average filtering time for each one of the different clustering thresholds is shown in Figure 6.12. We observe that after a threshold value of around 0.4, the filtering time shows a slight increase. As we move to even higher values of c, the increase in filtering time is even sharper. This can be explained as follows. At high values of c, sets of words with a high clustering ratio are also moved. These sets usually contain words that are often in user queries, and thus create sub-tries that store a large number of such sets. Moving highly clustered queries results in the perturbation of large tries that need to re-cluster, probably in a position with a lower clustering ratio. This way, by moving some already highly clustered queries to an even better position, we disturb the clustering of many other queries that subsequently cluster at a position with lower clustering ratio. Based on the results shown in Figure 6.12, we chose 0.4 as the clustering threshold to be used in the subsequent experiments.

Performance at Filtering Time

The second experiment targets the performance of filtering for different sizes of the query database after ReTrie is run. In this experiment we populated the query database db with different numbers of queries (ranging from 500K to 3M) using



Parameter	Value for set S_1	Value for set S_2
W	6702	6869
W_d	1004	1115
N	0.5M-3M	0.5M-3M
D	100	100
m	2.5%	2%
c	0.4, 0.6	0.4, 0.6

Figure 6.13: Performance of algorithm ReTrie for different clustering thresholds and sets of documents

algorithm BestFitTrie. We then used algorithm ReTrie with clustering threshold 0.4 to reorganise the queries stored in the database db to get a new database db'. We then chose one hundred documents from the NN corpus and used them as incoming documents for both databases db and db' of different sizes.

To conduct this experiment we choose two different sets of documents, namely S_1 and S_2, that contain one hundred documents each. Set S_1 consists of documents from the NN corpus, chosen to contain many words that are also contained in the under-clustered queries that algorithm ReTrie has chosen to move. This set is used to demonstrate the performance gain of algorithm ReTrie at times where similar documents arrive at high rates, e.g. imagine a scenario where the above algorithms are used in a news alert system and a sudden crisis occurs (e.g., an earthquake or terrorist act). Set S_2 is a randomly chosen set of documents from the NN corpus. This set is used to demonstrate the performance of the algorithm in the standard setting used to conduct the rest of the experiments.

Figure 6.13 shows the performance of ReTrie for two clustering thresholds ($c = 0.4$ and $c = 0.6$) and also in comparison with BestFitTrie. Initially one can observe that the results of the experiments in Section 6.4.6 and also our choice of 0.4 as a value for c are verified. The filtering performance of algorithm ReTrie for $c = 0.6$ is worse not only from its counterpart with $c = 0.4$, but also from algorithm BestFitTrie. The reasons for this are explained in the previous section. The performance of the algorithms is also affected by the document set used as publications. We can see that ReTrie is significantly better than BestFitTrie in cases where many simi-

lar documents get published at a short time interval, whereas the gain in filtering performance is small when a random document set is used. This can be explained as follows. In the case of random documents the improvement in clustering that is achieved by ReTrie is not exploited since not many of the re-clustered sub-tries are used, due to the statistical properties of the document set (many diverse subjects, larger vocabulary, etc.). On the other hand, in the case of similar documents, the clustering improvement leads to a performance benefit since the optimised sub-tries are used more often and lookups in $OT(d)$ are reduced. In general, from the set of experiments we conducted with document sets of different statistical properties, we observed improvement in filtering performance ranging from 2% to 18% for clustering thresholds ranging from 0.1 to 0.4. The time needed to re-organise the under-clustered sets of words was marginally higher compared to the update time shown in the experiments of Section 6.4.4 times the number of set of words that had to be moved.

Space Requirements

ReTrie needs about 22MB more memory than the rest of the trie-based algorithms for a database of 3M queries, due to the size of CA. This amount, although small compared with today's main memory capacities, is about one third of the total memory requirements for the rest of the trie-based algorithms. This increase in memory usage is the cost for the faster filtering times achieved by ReTrie.

6.4.7 Summary of Results

The experiments conducted in this section show the strengths and weaknesses of all algorithms. When no frequency information of word occurrences in documents is available, BestFitTrie seems to perform as much as 20% faster than prior work in the literature (achieving also a significantly higher throughput rate). Additionally BestFitTrie seems to remain relatively unaffected to the increase in document size, compared to the rest of the algorithms. This improvement comes at the cost of a small constant increase in storage cost compared to brute force approach, and a

small increase in query insertion time, which is a standard tradeoff in such a setting. Additionally BestFitTrie presents a sensitivity in the increase of the matching percentage, compared to the rest of the algorithms, but experiments show that it remains significantly faster despite this increase.

When word frequency information is available, variations of the original algorithms (namely BestFitTrie-irank and PrefixTrie-irank) are reported to perform faster. A new algorithm introduced, called LCWTrie, performs slightly better that the previous algorithms, with no extra storage cost associated. Finally, algorithm ReTrie seems an interesting addition to the algorithms presented so far. While the rest of the algorithms give a greedy static solution to the problem of query database organisation, ReTrie considers periodic re-organisations of the database to achieve better performance at filtering time. Experiments show that the parameters regulating this re-organisation process (e.g., the clustering threshold) should be carefully tuned and tailored to the needs of the specific scenario. Should this be done, further improvements of around 15% compared to BestFitTrie are reported.

6.5 Indexing Using Tries

In this section we discuss related research which is specific to this chapter and was not covered in Chapter 2. Initially we give a brief background on the trie data structure and then we present the trie variations that we extended to create our local indexing mechanisms.

The concept of tries was initially conceived by de la Briandais in [67], but the actual term was coined in [83] by Fredkin, who derived the naming from the term re*trie*val used in information retrieval systems. Tries are widely used in a number of different application domains ranging from dictionary management [14, 21, 119] and text compression [32] to natural language processing [23, 157], pattern matching [79, 172], IP routing [152] or searching for reserved words in a compiler [13]. The broad applicability of tries has resulted in considering them a general-purpose data structure with properties that are now well-understood due to a series of studies e.g., [72, 78, 107, 119, 167, 172].

There are several ways to implement a trie node depending on the application in mind, but the two most common ways are using arrays in the size of the alphabet (array tries) [83] and using lists with non-empty elements as roots of subtrees (list tries) [67, 118]. Array tries are better suited when the alphabet is relatively small, whereas the list implementation is best suited for large alphabets or trie nodes with few children, as opposed to a fixed size array consisting mainly of null pointers.

There are generally two ways to reduce the size of a trie, reducing the size of each one of the nodes, and reducing the number of nodes needed to represent a set of strings. Compact tries [185] are variants that are used to reduce the number of nodes needed to represent a certain string, by compacting chains of nodes that lead to a leaf without branching to a single node. Another idea for size reduction in a trie is to view the indexed strings as a set, rather than as a sequence. In this way the size of the resulting trie can be influenced, leading to the smallest trie. However, Comer showed that the problem of determining the smallest trie is NP-complete [60], thus several works have proposed heuristics to minimise a static trie (e.g., the O-Trie [59]).

In the approach presented in this chapter we extended the concept of a list trie to implement the data structures used to store user queries in main memory. The algorithm BestFitTrie (and its variations LCWTrie and ReTrie) utilise variations of list tries and techniques from compact tries to allow for the late creation of trie nodes in order to explore commonalities between user queries.

6.6 Conclusions

In this chapter we have presented efficient main-memory algorithms suitable for large scale information filtering with queries supporting Boolean and proximity operations on attribute values. Our work extends the SIFT framework to the $\mathcal{AWP}$ model, and presents new data structures and algorithms to efficiently filter incoming documents. Specifically, we improve current algorithms for this class of queries by introducing three new algorithms (namely BestFitTrie, LCWTrie, and ReTrie) and their variations. The efficiency of the new algorithms is investigated using an existing scientific collection and a synthetic query collection. A byproduct of our work is a

new corpus of documents and user queries which is representative of digital library scenarios, and can also be used by other researchers in the area of information filtering.

This was the last chapter presenting results of this thesis. In the next chapter we highlight the main achievements of our work, discuss possible directions for future work and conclude this thesis.

Chapter 7

Conclusions

To conclude this work we will present a short summary of the research conducted in this thesis, we will highlight our main contributions and provide possible directions for future research.

7.1 Summary

In this thesis we put our focus on the problem of distributed resource sharing in wide-area networks such as the Internet and the Web. In the architecture that we have proposed, each peer owns resources that it is willing to share and supports two kinds of basic functionality: information retrieval and information filtering. This functionality is unified in a single framework and a P2P architecture is adopted.

In our approach, nodes can implement any of the following types of services: super-peer service, provider service and client service. Nodes implementing the super-peer service (super-peers) form the message routing layer of the network. Each super-peer is responsible for serving a fraction of the clients by storing documents, indexing continuous queries, matching them against incoming (published) documents and creating notifications. The super-peers run a DHT protocol which is an extension of Chord. A node implementing the client service (client) connects to the network through a single super-peer node, which is its access point. Clients can connect, disconnect or even leave the system silently at any time. Clients are information consumers: they can pose one-time queries to receive relevant resources,

subscribe to resource publications with continuous queries and receive notifications about published resources (e.g., documents) that match their interests. Finally, the provider service (provider) is implemented by information sources that want to expose their contents to the clients of the system.

The main focus of this work was to provide models and languages for expressing publications and subscriptions, protocols that regulate super-peer interactions and query indexing mechanisms that are utilized by each one of the super-peers locally.

Publications and subscriptions in our architecture were expressed using a well-understood attribute-value model, called $\mathcal{AWPS}$, that is based on named attributes with free text as value interpreted under the Boolean and VSM (or LSI) models. Utilising a different data model (e.g., such as XML) and query language (e.g., such as XPath) would force us to change our distributed protocols for resource publication and query subscription, and also our local indexing algorithms to accommodate the new features introduced by the different model.

In the context of this work, we showed how to provide IR and IF functionality by using an extension of the Chord DHT. To achieve this, we have designed and implemented a set of protocols, collectively called DHTrie, that extend the Chord protocols assuming that publications and subscriptions are expressed in the model $\mathcal{AWPS}$. Since probability distributions associated with publication and query elements are expected to be skewed in such a scenario, achieving a balanced load among the nodes becomes an important problem. Thus, we studied important cases of load balancing for DHTrie and presented a new algorithm, based on the idea of load-shedding, which is also applicable to the standard DHT lookup problem.

In the architecture described above, clients subscribe to their access points with profiles that express their information needs, and providers expose their content using an appropriate meta-data model. Each super-peer is responsible for storing the queries, so that whenever a resource is published, the continuous queries satisfying it are found and notifications are sent to the appropriate clients. A facet of this work deals with the filtering problem that needs to be solved efficiently by each super-peer. We have proposed data structures and indexing algorithms that enable us to solve the filtering problem efficiently for large databases of queries expressed

in the model $\mathcal{AWP}$. The main idea behind these algorithms is to store sets of words compactly by exploiting their common elements. This is done using a variation of the trie data structure and appropriate algorithms are used for reorganising the query database when clustering of queries drops.

7.2 Contributions

In this section we summarise the primary contributions of this thesis. We have looked into the problem of offering distributed resource sharing functionality in a P2P environment and presented an architecture that unifies information retrieval and filtering in a single framework, while providing centralised and distributed algorithms that outperform the best algorithms in the literature. Our ultimate goal was to combine the scalability and efficiency of a distributed system, with the effectiveness and accuracy of a centralised system.

Specifically, we presented three IR-based models, studied their theory and answered questions related to satisfiability and entailment. In this context, we used methods from logic and complexity to show that the satisfiability problem for queries in $\mathcal{WP}$ and $\mathcal{AWP}$ is $\mathcal{NP}$-complete and the entailment problem is co$\mathcal{NP}$-complete and discussed cases where these problems can be solved in polynomial time.

Contrary to other approaches in the area, we aimed for exact query answering by extending the Chord protocols with IR and pub/sub functionality. In this way, we provided a distributed architecture with low network traffic and low publication latency at the same time. We also demonstrated the effectiveness of additional routing information, based on local interactions, in lowering message traffic and latency.

Finally, we proposed a solution for the local filtering problem that outperforms current literature. Our trie-based query indexing algorithms are 20% faster than their counterparts, offering sophisticated clustering of user queries and mechanisms for the adaptive reorganisation of the query database when filtering performance drops.

7.3 Open Problems

Let us now present a list of open problems that are directly related to the results of this work. Note that this list is meant to illustrate our possible future directions and it is not by any means exhaustive. We are primarily interested in supporting rich query languages in distributed information management systems and in designing distributed and centralised query and document indexing mechanisms that enable one-time and pub/sub functionality. However, issues such as security, privacy, locality, failure resilience and recovery are equally important issues in the design of such a system, and should not be neglected.

7.3.1 Rich Query Languages

In this work we presented three progressively more expressive data models and their respective query languages, and showed how to utilise them in a distributed resource sharing scenario. We would like to move to the utilisation of data models based on XML and queries based on the XQuery/XPath enriched with IR features such as phrases, word proximity, similarity, etc. It is also interesting to use the lessons learned in Chapter 3 to study the complexity of query evaluation in RDBMS with text functionalities, combinations of RDBMS and IR systems [52], and proposals for full-text extensions to XML [20].

7.3.2 Approximate Information Filtering

The problem of information filtering has lately received considerable attention from various research communities including researchers from information retrieval, databases, distributed computing, digital libraries, agent systems [46, 187, 203] and others. All the approaches taken so far (including the one presented in the context of this thesis) have the underlying hypothesis that the subscriber is interested in *all* the events that were published in the network. It would also be interesting to support *approximate information filtering* functionality. In this context, a user subscribes with a continuous query and monitors *some* (namely the most interesting) sources of the network. The user query is replicated to these sources and only incoming

documents *from these sources* are forwarded to him. The system is responsible for managing the user query, discover new potential sources and move queries to better ones. This approach resembles centralised approaches currently taken for filtering news items, based on a profile of user preferences [62]. In these approaches, however, the emphasis is on duplicate elimination whereas in our case is on information quality, scalability and efficiency.

7.3.3 Load Balancing

In the DHT literature, work on load balancing has concentrated on two particular problems: *address-space load balancing* and *item load balancing*. The former problem is how to partition the address-space of a DHT "evenly" among keys; it is typically solved by relying on consistent hashing and constructions such as virtual servers [180] or potential nodes [113]. In the latter problem, load has to be balanced in the presence of data items with arbitrary load distributions [4, 113] as in our case. The load balancing algorithm presented in this thesis is based on the concept of load shedding to ensure uniform load distribution. This algorithm is also applicable to the standard DHT look-up problem, and many questions regarding the details and the behavior of the algorithm remain open:

- Which peers should be contacted when a load-shedding decision is made from an overloaded peer P? Normally one would have to contact all the peers in the network to notify them about this change, but we conjecture that notifying only the nodes that contact P more often will suffice in most cases.

- Load-stealing (or load-acquisition) is also a promising technique. What are the cases where it is more effective that load-shedding? What are the different parameters that affect the behaviour of the two approaches?

- Different load distributions often require different approaches. How can we identify the load balancing algorithms appropriate for each type of skew?

- Load balancing plays an important role in the scalability and efficiency of any P2P system, especially in structured ones, where failures are more expensive.

It is therefore necessary to integrate load balancing mechanisms at the level of the DHT infrastructure, which is not yet clear how to achieve even given works such as [4, 113, 113, 180].

References

[1] K. Aberer. P-Grid: A Self-Organizing Access Structure for P2P Information Systems. In *Conference on Cooperative Information Systems (CoopIS)*, pages 179–194, 2001.

[2] K. Aberer and J. Wu. A Framework for Decentralized Ranking in Web Information Retrieval. In *Proceedings of Asia-Pacific Web Conference (APWeb)*, pages 213–226, 2003.

[3] K. Aberer, M. Punceva, M. Hauswirth, and R. Schmidt. Improving Data Access in P2P Systems. *IEEE Internet Computing*, 6(1):58–67, 2002.

[4] K. Aberer, A. Datta, and M. Hauswirth. Multifaceted Simultaneous Load Balancing in DHT-based P2P systems: A new game with old balls and bins. Technical Report IC/2004/23, EPFL, 2004.

[5] K. Aberer, F. Klemm, M. Rajman, and J. Wu. An Architecture for Peer-to-Peer Information Retrieval. In *Proceedings of the International Workshop on Peer-to-Peer Information Retrieval (P2PIR)*, Sheffield, UK, July 2004.

[6] K. Aberer, L. Onana Alima, A. Ghodsi, S. Girdzijauskas, M. Hauswirth, and S. Haridi. The essence of P2P: A reference architecture for overlay networks. In *Proccedings of the International Conference on Peer-to-Peer Computing (P2P)*, August 2005.

[7] K. Aberer, A. Datta, M. Hauswirth, and R. Schmidt. Indexing Data-oriented Overlay Networks. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 685–696, Trondheim, Norway, August 2005.

[8] Karl Aberer, Phillipe Cudre-Mauroux, and Manfred Hauswirth. The Chatty Web: Emergent Semanics Through Gossiping. In *Proceedings of the Interna-*

tional World Wide Web Conference (WWW), Budapest, Hungary, 20-24 May 2003. ACM.

[9] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[10] I. Aekaterinidis and P. Triantafillou. Internet scale string attribute publish/subscribe data networks. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, pages 44–51, 2005.

[11] I. Aekaterinidis and P. Triantafillou. PastryStrings: A Comprehensive Content-Based Publish/Subscribe DHT Network. In *Proceedings of the International Conference on Distributed Computing and Systems (ICDCS)*, July 2006.

[12] M. K. Aguilera, R. E. Strom, D.C. Sturman, M. Astley, and T.D. Chandra. Matching Events in a Content-based Subscription System. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 53–62, New York, May 1999. Association for Computing Machinery.

[13] A. V. Aho, R. Sethi, and J.D. Ullman. *Compilers, Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[14] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, Massachusetts, 1983.

[15] L.O. Alima, S. El-Ansary, P. Brand, and S. Haridi. DKS(N, k, f): A Family of Low Communication, Scalable and Fault-Tolerant Infrastructures for P2P Applications. In *Cluster Computing and the Grid (CCGRID)*, pages 344–350, 2003.

[16] L.O. Alima, A. Ghodsi, P. Brand, and S. Haridi. Multicast in DKS(N, k, f) Overlay Networks. In *Proceedings of the International Conference on Principles of DIstributed Systems (OPODIS)*, 2003.

[17] L.O. Alima, A. Ghodsi, and S. Haridi. A Framework for Structured Peer-to-Peer Overlay Networks. In *Global Computing 2004*, volume 3267 of *LNCS*, pages 223–250, 2004.

[18] M. Altinel and M.J. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2000.

[19] M. Altinel, D. Aksoy, T. Baby, M. Franklin, W. Shapiro, and S. Zdonik. DBIS-toolkit: Adaptable Middleware for Large-scale Data Delivery. In *Proceedings of the ACM SIGMOD Conference*, 1999.

[20] S. Amer-Yahia, C. Botev, and J. Shanmugasundaram. TeXQuery: a full-text search extension to Query. In *Proceedings of the World Wide Web Conference (WWW)*, pages 583–594. ACM Press, 2004.

[21] J.-I. Aoe, K. Morimoto, and T. Sato. An Efficient Implementation of Trie Structures. *Software–Practice and Experience*, 22(9):695–721, 1992.

[22] J. Aspnes and G. Shah. Skip Graphs. In *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384–393, Baltimore, MD, USA, 12–14 January 2003.

[23] R. Baeza-Yates and G. Gonnet. Fast Text Searching for Regular Expressions or Automaton Simulation on Tries. *Journal of the ACM*, 43(6):915–936, 1996.

[24] R.A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[25] H. Balakrishnan, M.F. Kaashoek, D.R. Karger, R. Morris, and I. Stoica. Looking up data in P2P systems. *Communications of the ACM*, 46(2):43–48, 2003.

[26] H. Balakrishnan, S. Shenker, and M. Walfish. Semantic-Free Referencing in Linked Distributed Systems. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[27] W.T. Balke. *Peer-to-Peer Systems and Applications*, chapter Information Retrieval in Peer-to-Peer Systems. Number 3485 in LNCS. Springer, 2005.

[28] W.T. Balke, W. Nejdl, W. Siberski, and U. Thaden. DL Meets P2P - Distributed Document Retrieval Based on Classification and Content. In *Proceedings of the European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, pages 379–390, 2005.

[29] W.T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive Distributed Top k Retrieval in Peer-to-Peer Networks. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 174–185, 2005.

[30] N.J. Belkin and W.B. Croft. Information Filtering and Information Retrieval: Two Sides of the Same Coin? *Communications of the ACM*, 35(12):29–38, 1992.

[31] T.A.H. Bell and A. Moffat. The Design of a High Performance Information Filtering System. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 12–20, 1996.

[32] T.C. Bell, J.G. Cleary, and I.H. Witten. Text Compression. *Prentice-Hall*, 1990.

[33] M. Bender, S. Michel, G. Weikum, and C. Zimmer. Bookmark-Driven Query Routing in Peer-to-Peer Web Search. In *Proceedings of the International Workshop on Peer-to-Peer Information Retrieval (P2PIR)*, 2004.

[34] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Improving Collection Selection with Overlap-Awareness. In Ricardo A. Baeza-Yates, Nivio Ziviani, Gary Marchionini, Alistair Moffat, and John Tait, editors, *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 67–74, Salvador, Brazil, 2005. ACM.

[35] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. MINERVA: Collaborative P2P Search (Demo). In Klemens Bhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 1263–1266, Trondheim, Norway, 2005. ACM.

[36] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1987.

[37] A. Bharambe, S. Rao, and S. Seshan. Mercury: A Scalable Publish-Subscribe System for Internet Games. In *Proceedings of the International Workshop on Network and System Support for Games (Netgames)*, Braunchweig, Germany, 2002.

[38] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of ACM SIGCOMM Conference*, Portland, Oregon, USA, 2004.

[39] S. Brin and L. Page. The anatomy of a large–scale hypertextual Web search engine. In *Proceedings of the World Wide Web Conference (WWW)*, 1998.

[40] J.P. Callan. Learning While Filtering Focuments. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 224–231, 1998.

[41] J.P. Callan. Document Filtering With Inference Networks. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 1996.

[42] J.P. Callan, W.B. Croft, and S.M. Harding. The INQUERY retrieval system. In *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA)*, pages 78–83. Springer-Verlag, 1992.

[43] A. Campialla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient Filtering in Publish Subscribe Systems Using Binary Decision Diagrams. In *Proceedings of the International Conference on Software Engeneering (ICSE)*, pages 443–452, Los Alamitos, California, May12–19 2001. IEEE Computer Society.

[44] A. Carzaniga, D.S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an Internet-scale event notification service. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 219–227, 2000.

[45] A. Carzaniga, D.-S. Rosenblum, and A.L Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332–383, 2001.

[46] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332–383, August 2001.

[47] U. Cetintemel, M.J. Franklin, and C.L. Giles. Self-Adaptive User Profiles for Large-Scale Data Delivery. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 622–633, 2000.

[48] C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient Filtering of XML Documents with XPath Expressions. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 235–244, February 2002.

[49] C.-C. K. Chang, H. Garcia-Molina, and A. Paepcke. Predicate Rewriting for Translating Boolean Queries in a Heterogeneous Information System. *ACM Transactions on Information Systems (TOIS)*, 17(1):1–39, 1999.

[50] C.-C.K. Chang. *Query and Data Mapping Across Heterogeneous Information Sources*. PhD thesis, Stanford University, January 2001.

[51] C.-C.K. Chang, H. Garcia-Molina, and A. Paepcke. Boolean Query Mapping across Heterogeneous Information Sources. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 8(4):515–521, 1996.

[52] S. Chaudhuri, R. Ramakrishnan, and G. Weikum. Integrating DB and IR Technologies: What is the Sound of One Hand Clapping? In *Proccedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 1–12, 2005.

[53] T.T. Chinenyanga and N. Kushmerick. Expressive retrieval from XML documents. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, September 2001.

[54] P. A. Chirita, S. Idreos, M. Koubarakis, and W. Nejdl. Publish/Subscribe for RDF-based P2P Networks. In *Proceedings of the European Semantic Web Symposium (ESWS)*, 2004.

[55] P.A. Chirita, W. Nejdl, and O. Scurtu. Knowing Where to Search: Personalized Search Strategies for Peers in P2P Networks. In *Proceedings of the International Workshop on Peer-to-Peer Information Retrieval (P2PIR)*, 2004.

[56] I. Clarke, T.W. Hong, S.G. Miller, O. Sandberg, and B. Wiley. Protecting Free Expression Online with Freenet. *IEEE Internet Computing*, 6(1):40–49, January 2002.

[57] E. Cohen, A. Fiat, and H. Kaplan. Associative Search in Peer to Peer Networks: Harnessing Latent Semantics. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2003.

[58] W.W. Cohen. WHIRL: A word-based information representation language. *Artificial Intelligence*, 118(1-2):163–196, 2000.

[59] D. Comer. Analysis of a Heuristic for Trie Minimization. *ACM Transactions on Database Systems (TODS)*, 6(3):513–537, September 1981.

[60] D. Comer and R. Sethi. The Complexity of Trie Index Construction. *Journal of the ACM*, 24(3):428–440, 1977.

[61] G.M. Del Corso, A. Gulli, and F. Romani. Ranking a stream of news. In *Proceedings of the World Wide Web Conference (WWW)*, pages 97–106, 2005.

[62] G.M.d. Corso, A. Gulli, and F. Romani. Ranking a Stream of News. In *Proceedings of the World Wide Web Conference (WWW)*, 2005.

[63] A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. In *Proceedings of the 28th International Conference on Distributed Systems*, July 2002.

[64] F.M. Cuenca-Acuna and T.D. Nguyen. Text-Based Content Search and Retrieval in Ad-hoc P2P Communities. In *Proceedings of the Networking 2002 Workshops*, number 2376 in LNCS, pages 220–234. Springer-Verlag, 2002.

[65] A. Datta, M. Hauswirth, R. John, R. Schmidt, and K. Aberer. Range Queries in Trie-Structured Overlays. In *Proccedings of the International Conference on Peer-to-Peer Computing (P2P)*, pages 57–66, 2005.

[66] N. de Bruijn. A Combinatorial Problem. In *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen*, volume 49, pages 758–764, 1946.

[67] R. de la Briandais. File searching using variable length keys. In *Proceedings of the Western Joint Computer Conference*, pages 295–298, 1959.

[68] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[69] R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49(1-3):61–95, 1991. Special volume on Knowledge Representation.

[70] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Management. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–12, August 1987.

[71] P.J. Denning. Electronic Junk. *Communications of the ACM*, 25(3):163–165, 1982.

[72] L. Devroye. A study of trie-like structures under the density model. *Annals of Applied Probability*, 2(2):402–434, 1992.

[73] Y. Diao, M. Altinel, M.J. Franklin, H. Zhang, and P. Fischer. Path Sharing and Predicate Evaluation for High-Performance XML Filtering. *ACM Transactions on Database Systems (TODS)*, 2003.

[74] L. Dong. Automatic term extraction and similarity assessment in a domain specific document corpus. Master's thesis, Department of Computer Science, Dalhousie University, Halifax, Canada, 2002.

[75] F. Fabret, H.A. Jacobsen, F. Llirbat, J. Pereira, K.A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proceedings of the ACM SIGMOD Conference*, 2001.

[76] D. Faensen, L. Faulstich, H. Schweppe, A. Hinze, and A. Steidinger. Hermes – A Notification Service for Digital Libraries. In *Proceedings of the Joint ACM/IEEE Conference on Digital Libraries (JCDL)*, Roanoke, Virginia, USA, 2001.

[77] A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. In *Symposium on Discrete Algorithms (SODA)*, pages 94–103, 2002.

[78] P. Flajolet. On the Performance Evaluation of Extendible Hashing and Trie Searching. *Acta Informatica*, 20:345–369, 1983.

[79] P. Flajolet and C. Puech. Partial match retrieval of multidimensional data. *Journal of the ACM*, 33(2):371–407, 1986.

[80] P.W. Foltz and S.T. Dumais. Personalized Information Delivery: An Analysis of Information Filtering Methods. *Communications of the ACM*, 35(12), 1992.

[81] M.J. Franklin and S.B. Zdonik. "Data In Your Face": Push Technology in Perspective. In *Proceedings of the ACM SIGMOD Conference*, pages 516–519, 1998.

[82] K. Frantzi, S. Ananiadou, and H. Mima. Automatic recognition of multi-word terms: the C-value/NC-value method. *International Journal on Digital Libraries*, 5(2), 2000.

[83] E. Fredkin. Trie Memory. *Communications of the ACM*, 3(9):490–499, 1960.

[84] M.J. Freedman and R. Vingralek. Efficient Peer-to-Peer Lookup Based on a Distributed Trie. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 1 of *LNCS*, 2002.

[85] N. Fuhr and K. Großjohann. XIRQL: An XML Query Language Based on Information Retrieval Concepts. *ACM Transactions on Information Systems (TOIS)*, 22(2):313–356, April 2004.

[86] E. Gabrilovich, S.T. Dumais, and E. Horvitz. Newsjunkie: providing personalized newsfeeds via analysis of information novelty. In *Proceedings of the World Wide Web Conference (WWW)*, pages 482–490, 2004.

[87] P. Ganesan, Q. Sun, and H. Garcia-Molina. YAPPERS: A Peer-to-Peer Lookup Service over Arbitrary Topology. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2003.

[88] P. Ganesan, P.K. Gummadi, and H. Garcia-Molina. Canon in G Major: Designing DHTs with Hierarchical Structure. In *Proceedings of the International Conference on Distributed Computing and Systems (ICDCS)*, pages 263–272, 2004.

[89] H. Garcia-Molina. Peer-to-Peer Data Management. Powerpoint slides of keynote address given at ICDE, 2002. Available from `http://www-db.stanford.edu/peers/`.

[90] B. Gedik and L. Liu. PeerCQ: A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System. In *Proceedings of the IEEE International Conference on Distributed Computer Systems (ICDCS)*, May 2003.

[91] V. Gopalakrishnan, B. Bhattacharjee, and P. Keleher. Distributing Google. In *Proceedings of the IEEE International Workshop on Networking Meets Databases (NetDB)*, April 2006.

[92] T.J. Green, G. Miklau, M. Onizuka, and D. Suciu. Processing XML Streams with Deterministic Automata. In *Proceedings of the Conference on Database Theory (ICDT)*, pages 173–189, Siena, Italy, January 2003.

[93] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: Content-Based Publish/Subscribe over P2P Networks. In *Proceedings of the International Conference on Distributed Systems Paltforms (Middleware)*, Toronto, Ontario, Canada, October 18-22 .

[94] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an Efficient and Stable P2P DHT through Increased Memory and Background Overhead. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2 of *LNCS*, 2003.

[95] M. Harren, J.M. Hellerstein, R. Huebsch, T. Loo, S. Shenker, and I. Stoica. Complex Queries in DHT-based Peer-to-Peer Networks. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[96] N.J.A. Harvey, M.B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[97] K. Hildrum, J. Kubiatowicz, S. Rao, and B.Y. Zhao. Distributed object location in a dynamic network. In *Proceedings of the ACM Symposium on Parallel Algorithms (SPAA)*, pages 41–52, 2002.

[98] K. Hildrum, J.D. Kubiatowicz amd S. Rao, and B.Y. Zhao. Distributed object location in a dynamic network. *Theory of Computing Systems*, 37(3):405–440, 2004.

[99] H.J.Siegel. Interconnection Networks for SIMD Machines. *Computer*, 12(6): 57–65, 1979.

[100] H.-C. Hsiao and C.-T. King. Similarity Discovery in Structured P2P Overlays. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, 2003.

[101] H.-C. Hsiao and C.-T. King. Tornado: a capability-aware peer-to-peer storage overlay. *Journal of Parallel and Distributed Computing (JPDC)*, 64(6):747–758, 2004.

[102] R. Huebsch. Content-Based Multicast: Comparison of Implementation Options. Technical Report UCB//CSD-03-1229, UC Berkeley, February 2003.

[103] D.A. Hull, J.O. Pedersen, and H. Schütze. Method Combination For Document Filtering. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 279–287, 1996.

[104] Stratos Idreos, Manolis Koubarakis, and Christos Tryfonopoulos. P2P-DIET: One-Time and Continuous Queries in Super-Peer Networks. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT)*, pages 851–853, Heraklion, Greece, March 2004.

[105] Stratos Idreos, Manolis Koubarakis, and Christos Tryfonopoulos. P2P-DIET: An Extensible P2P Service that Unifies Ad-hoc and Continuous Querying in Super-Peer Networks. In *Proceedings of the ACM SIGMOD Conference*, pages 933–934, Paris, France, June 2004.

[106] Stratos Idreos, Christos Tryfonopoulos, Manolis Koubarakis, and Yannis Drougas. Query Processing in Super-Peer Networks with Languages Based on Information Retrieval: the P2P-DIET Approach. In *Proceedings of the International Workshop on Peer-to-peer Computing and Databases (P2P&DB)*, Herakion, Greece, March 2004.

[107] P. Jacquet and W. Szpankowski. Analysis of digital tries with Markovian dependency. *IEEE Transactions on Information Theory (TOIT)*, 37(5):1470–1475, 1991.

[108] M. Jelasity and O. Babaoglu. T-Man: Gossip-Based Overlay Topology Management. In *Proceedings of the International Workshop on Engineering Self-Organising Applications (ESOA)*, volume 3910 of *LNCS*, Berlin, Germany, 2006. Springer-Verlag.

[109] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations. In *Proceedings of the International Conference on Distributed Systems Paltforms (Middleware)*, Toronto, Canada, October 2004.

[110] G.P. Jesi, A. Montresor, and O. Babaoglu. Proximity-Aware Superpeer Overlay Topologies. In *Proceedings of the International Workshop on Self-Managed Systems & Services (SelfMan'06)*, Dublin, Ireland, June 2006. Springer-Verlag.

[111] M.F. Kaashoek and D.R. Karger. Koorde: A Simple Degree-Optimal Distributed Hash Table. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2 of *LNCS*, 2003.

[112] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *ACM Symposium on Theory of Computing (STOC)*, 1997.

[113] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *Proceedings of the ACM Symposium on Parallel Algorithms (SPAA)*, 2004.

[114] I.A. Klampanos and J.M. Jose. An architecture for peer-to-peer information retrieval. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 401–402, 2003.

[115] I.A. Klampanos and J.M. Jose. An Architecture for Information Retrieval over Semi-Collaborating Peer-to-Peer Networks. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, 2004.

[116] J. Kleinberg. Authoratitive sources in a hyperlinked environment. *Journal of the ACM*, 48(5):604–632, 1999.

[117] F. Klemm and K. Aberer. Aggregation of a Term Vocabulary for Peer-to-Peer Information Retrieval: a DHT Stress Test. In *Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, Trondheim, Norway, August 2005.

[118] D.E. Knuth. *The Art of Computer Programming*, volume 1: Fundamental Algorithms. Addison-Wesley, Reading, Massachusetts, 1973.

[119] D.E. Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching. Addison-Wesley, Reading, Massachusetts, 1973.

[120] G. Koloniari and E. Pitoura. Content-Based Routing of Path Queries in Peer-to-Peer Systems. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 2004.

[121] G. Koloniari and E. Pitoura. Peer-to-Peer Management of XML Data: Issues and Research Challenges. *Sigmod Record*, June 2005.

[122] M. Koubarakis. The Complexity of Query Evaluation in Indefinite Temporal Constraint Databases. *Theoretical Computer Science*, 171:25–60, January 1997. Special Issue on Uncertainty in Databases and Deductive Systems, Editor: L.V.S. Lakshmanan.

[123] Manolis Koubarakis, Theodoros Koutris, Christos Tryfonopoulos, and Paraskevi Raftopoulou. Information Alert in Distributed Digital Libraries: The Models, Languages, and Architecture of DIAS. In *Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, pages 527–542, Rome, Italy, September 2002.

[124] Manolis Koubarakis, Christos Tryfonopoulos, Paraskevi Raftopoulou, and Theodoros Koutris. Data Models and Languages for Agent-Based Textual Information Dissemination. In *Proceedings of the 6th International Workshop on Cooperative Information Agents (CIA)*, volume 2446 of *Lecture Notes in Artificial Intelligence*, pages 179–193, Madrid, Spain, September 2002. Springer.

[125] Manolis Koubarakis, Christos Tryfonopoulos, Stratos Idreos, and Yannis Drougas. Selective Information Dissemination in P2P Networks: Problems

and Solutions. *SIGMOD Record, Special Issue on Peer-to-Peer Data Management*, 32(3):71–76, 2003.

[126] Manolis Koubarakis, Spiros Skiadopoulos, and Christos Tryfonopoulos. Logic and Computational Complexity for Boolean Information Retrieval. *IEEE Transactions on Knowledge and Data Engineering*, 2006. To appear.

[127] 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 *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2 of *LNCS*, 2003.

[128] W. Litwin. Linear Hashing: A New Tool For File And Table Addressing. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 212–223, 1980.

[129] W. Litwin, M.-A. Neimat, and D.A. Schneider. LH* - A Scalable, Distributed Data Structure. *ACM Transactions on Database Systems (TODS)*, 21(4):480–525, December 1996.

[130] J. Lu and J. Callan. Federated search of text-based digital libraries in hierarchical peer-to-peer networks. In *Proceedings of the European Conference on Information Retrieval Research (ECIR)*, 2005.

[131] J. Lu and J. Callan. Content-based retrieval in hybrid peer-to-peer networks. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, 2003.

[132] J. Lu and J. Callan. Merging retrieval results in hierarchical peer-to-peer networks. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 472–473, 2004.

[133] H.P. Luhn. A Business Intelligence System. *IBM Journal of Reasearch and Development*, 2(4):314–319, 1958.

[134] T. Luu, F. Klemm, M. Rajman, and K. Aberer. Using Highly Discriminative Keys for Indexing in a Peer-to-Peer Full-Text Retrieval System. Technical Report 2005041, Ecole Polytechnique Federale de Lausenne (EPFL), School of Computer and Communication Sciences, July 2005.

[135] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 183–192, 2002.

[136] G.S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed Hashing in a Small World. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

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

[138] S. Marti, P. Ganesan, and H. Garcia-Molina. SPROUT: P2P Routing with Social Networks. In *Proceedings of the International Workshop on Peer-to-Peer Computing and Databases (P2P&DB)*, pages 425–435, 2004.

[139] P. Maymounkov and D. Mazieres. PKademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 1 of *LNCS*, 2002.

[140] S. Michel, P. Triantafillou, and G. Weikum. KLEE: A Framework for Distributed Top-k Query Algorithms. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 637–648, 2005.

[141] S. Michel, P. Triantafillou, and G. Weikum. MINERVA$_\infty$: A Scalable Efficient Peer-to-Peer Search Engine. In Gustavo Alonso, editor, *Proceedings of the International Conference on Distributed Systems Paltforms (Middleware)*, volume 3790 of *Lecture Notes in Computer Science*, pages 60–81, Grenoble, France, 2005. Springer.

[142] S. Michel, M. Bender, P. Triantafillou, and G. Weikum. IQN Routing: Integrating Quality and Novelty in P2P Querying and Ranking. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, Munich, Germany, March 2006.

[143] E. Milios, Y. Zhang, B. He, and L. Dong. Automatic Term Extraction and Document Similarity in Special Text Corpora. In *Proceedings of the Pacific Association for Computational Linguistics Conference (PACLing)*, pages 275–284, Halifax, Canada, August 2003.

[144] A. Montresor, M. Jelasity, , and O. Babaoglu. Gossip-based Aggregation in Large Dynamic Networks. *ACM Transactions on Computer Systems (TOCS)*, 23(3):219–252, 2005.

[145] M. Morita and Y. Shinoda. Information Filtering Based on User Behaviour Analysis and Best Match Text Retrieval. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 272–281, 1994.

[146] W. Müller, M. Eisenhardt, and A. Henrich. Scalable summary based retrieval in P2P networks. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, pages 586–593, 2005.

[147] M. Naor and U. Wieder. A Simple Fault Tolerant Distributed Hash Table. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2 of *LNCS*, 2003.

[148] G. Navarro and R.A. Baeza-Yates. Proximal Nodes: A Model to Query Document Databases by Content and Structure. *ACM Transactions on Information Systems (TOIS)*, 15(4):400–435, 1997.

[149] W. Neidl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch. EDUTELLA: A P2P Networking Infrastructure Based on RDF. In *Proceedings of the International World Wide Web Conference (WWW)*. May 2002.

[150] W. Nejdl, W. Siberski, U. Thaden, and W.T. Balke. Top-k Query Evaluation for Schema-Based Peer-to-Peer Networks. In *Proceedings of the International Semantic Web Conference*, pages 137–151, 2004.

[151] B. Nguyen, S. Abiteboul, G.Cobena, and M. Preda. Monitoring XML Data on the Web. In *Proceedings of the ACM SIGMOD Conference*, Santa Barbara, CA, USA, 2001.

[152] S. Nilsson and G. Karlsson. IP-Address Lookup Using LC-Tries. *IEEE Journal on Selected Areas in Communication*, 17(6):1083–1092, 1999.

[153] National Institute of Standards and Technology. Secure hash standard, 1995. Publication 180-1.

[154] P2P-DIET home page. URL `http://www.www.intelligence.tuc.gr/p2pdiet/`.

[155] P2PIR. Workshop on Peer-to-Peer Information Retrieval, July 2004.

[156] P2PIR. Workshop on Peer-to-Peer Information Retrieval, November 2005.

[157] J.L. Peterson. Computer Programs for Detecting and Correcting Spelling Errors. *Communications of the ACM*, 23(12):676–686, 1980.

[158] Y. Petrakis and E. Pitoura. On Constructing Small Worlds in Unstructured Peer-to-Peer Systems. In *Proceedings of the International Workshop on Peer-to-Peer Computing and Databases (P2P&DB)*, Heraklion, Crete, Greece, March 2004.

[159] U. Pfeifer, N. Fuhr, and T. Huynh. Searching Structured Documents with the Enhanced Retrieval Functionality of freeWAIS-sf and SFgate. *Computer Networks and ISDN Systems*, 27(6):1027–1036, 1995.

[160] P.R. Pietzuch and J.M. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *Proceedings of the International Workshop on Distributed Event-Based Systems (DEBS)*, July 2002.

[161] T. Pitoura, N. Ntarmos, and P. Triantafillou. Replication, Load Balancing and Efficient Range Query Processing in DHTs. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, March 2006.

[162] C.G. Plaxton, R. Rajaraman, and A.W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. *Theory of Computing Systems*, 32(3):241–280, 1999.

[163] M.F. Porter. An Algorithm for Suffix Striping. *Program*, 14(3):130–137, 1980.

[164] P. Raftopoulou and E.G.M. Petrakis. Knowledge Sharing in Multi-Layer P2P Networks. In *Proceedings of IEEE STEP 4th International Workshop on Net-Centric Computing (NCC)*, Budapest, Hungary, 24-25 September 2005.

[165] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-addressable Network. In *Proceedings of ACM SIGCOMM Conference*, 2001.

[166] S. Ratnasamy, M. Handley, R.M. Karp, and S. Shenker. Application-Level Multicast Using Content-Addressable Networks. In *Proceedings of International Workshop on Networked Group Communication (NGC)*, pages 14–29, London, UK, November 2001.

[167] M. Regnier and P. Jacquet. New Results on the Size of Tries. *IEEE Transactions on Information Theory (TOIT)*, 35(1):203–205, 1989.

[168] M.E. Renda and J. Callan. The robustness of content-based search in hierarchical peer to peer networks. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, pages 562–570, 2004.

[169] P. Revesz. *Introduction to Constraint Databases*. Springer, 2002.

[170] P.Z. Revesz. A Closed Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints. *Theoretical Computer Science*, 116(1):117–149, 1993.

[171] P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. In *Proceedings of the International Conference on Distributed Systems Paltforms (Middleware)*, pages 21–40, 2003.

[172] Ronald L. Rivest. Partial-Match Retrieval Algorithms. *SIAM Journal on Computing*, 5(1):19–50, 1976.

[173] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralised Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the International Conference on Distributed Systems Paltforms (Middleware)*, November 2001.

[174] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The Design of a Large-scale Event Notification Infrastructure. In J. Crowcroft and M. Hofmann, editors, *Proceedings of the International Workshop COST264*, 2001.

[175] O.D. Sahin, F. Emekci, D. Agrawal, and A.E. Abbadi. Content-Based Similarity Search over Peer-to-Peer Systems. In *Proceedings of the International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P)*, 2004.

[176] K. Sankaralingam, S. Sethumadhavan, and J.C. Browne. Distributed Pagerank for P2P Systems. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 58–69, 2003.

[177] M.T. Schlosser, M. Sintek, S. Decker, and W. Nejdl. HyperCuP - Hypercubes, Ontologies, and Efficient Search on Peer-to-Peer Networks. In *Proceedings of the 1st Workshop on Agents and P2P Computing (AP2PC)*, pages 112–124, Bologna, Italy, 2002.

[178] S.M. Shi, J. Yu, G. Yang, and D.X. Wang. Distributed Page Ranking in Structured P2P Networks. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, pages 179–186, 2003.

[179] S. Srinivasan and E. Zegura. Network Measurement as a Cooperative Enterprise. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[180] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM Conference*, San Diego, California, August 2001.

[181] I. Stoica, D. Adkins, S. Ratnasamy, S. Shenker, S. Surana, and S. Zhuang. Internet Indirection Architecture. In *Proceedings of ACM SIGCOMM Conference*, pages 73–86, August 2002.

[182] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Frans Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking*, 11 (1):17–32, 2003.

[183] J. Stribling, I.G. Councill, J. Li, M.F. Kaashoek, D.R. Karger, R. Morris, and S. Shenker. OverCite: A Cooperative Digital Research Library. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 69–79, 2005.

[184] T. Suel, C. Mathur, J. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram. ODISSEA: A Peer-to-Peer Architecture for Scalable

Web Search and Information Retrieval. In *Proceedings of the International Workshop on the Web and Databases (WebDB)*, pages 67–72, 2003.

[185] E.H. Sussenguth. Use of Tree Structures for Processing Files. *Communications of the ACM*, 6(5):272–279, 1963.

[186] D. Tam, R. Azimi, and H.-Arno Jacobsen. Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables. In *Proceedings of the International Workshop On Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, September 2003.

[187] C. Tang and Z. Xu. pFilter: Global Information Filtering and Dissemination Using Structured Overlays. In *Proceedings of the International Workshop on Future Trends in Distributed Computing Systems (FTDCS)*, 2003.

[188] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information Retrieval in Structured Overlays. In *Proceedings of HotNets-I*, 2002.

[189] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of ACM SIGCOMM Conference*, 2003.

[190] W.W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A.P. Buchmann. A Peer-to-Peer Approach to Content-Based Publish/Subscribe. In *Proceedings of the International Workshop on Distributed Event-Based Systems (DEBS)*, June 2003.

[191] M. Theimer and M. Jones. Overlook: scalable name service on an overlay network. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 52–64, 2002.

[192] A. Theobald and G. Weikum. Adding Relevance to XML. In *Proceedings of the International Workshop on the Web and Databases (WebDB)*, pages 105–124, 2000.

[193] P. Triantafillou and A. Economides. Subscription Summarization: A New Paradigm for Efficient Publish/Subscribe Systems. In *Proceedings of the International Conference on Distributed Computing and Systems (ICDCS)*, Tokyo, Japan, March 2004.

[194] P. Triantafillou and T. Pitoura. Towards a Unifying Framework for Complex Query Processing over Structured Peer-to-Peer Data Networks. In *Proceedings of the 1st International Workshop On Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, September 2003.

[195] Christos Tryfonopoulos. Agent-Based Textual Information Dissemination: Data Models, Query Languages, Algorithms and Computational Complexity. Master's thesis, Department of Electronic and Computer Engineering, Technical University of Crete, Greece, October 2002.

[196] Christos Tryfonopoulos and Manolis Koubarakis. Implementing Publish/Subscribe Systems with Languages from Information Retrieval on Top of Structured Overlay Networks. In *Proceedings of the International Workshop on Peer-to-Peer and IR (P2P&IR)*, Sheffield, UK, July 2004.

[197] Christos Tryfonopoulos and Manolis Koubarakis. Distributed Resource Sharing using Self-Organized Peer-to-Peer Networks and Languages from Information Retrieval. In *Proceedings of the International Workshop on Self-* Properties in Complex Information Systems (Self-*)*, Bertinoro, Italy, May 2004.

[198] Christos Tryfonopoulos and Manolis Koubarakis. Selective Dissemination of Information in P2P Systems: Data Models, Query Languages, Algorithms and Computational Complexity. Technical Report TR-ISL-02-2003, Department of Electronic and Computer Engineering, Technical University of Crete, 2002.

[199] Christos Tryfonopoulos and Manolis Koubarakis. Publish/Subscribe Systems with Distributed Hash Tables and Languages from Information Retrieval. In *Proceedings of the 6th Workshop on Distributed Data and Structures (WDAS)*, Lausanne, Switzerland, July 2004.

[200] Christos Tryfonopoulos, Manolis Koubarakis, and Yannis Drougas. Filtering Algorithms for Information Retrieval Models with Named Attributes and Proximity Operators. In *Proceedings of the 27th Annual International ACM SIGIR Conference*, pages 313–320, Sheffield, UK, July 2004.

[201] Christos Tryfonopoulos, Stratos Idreos, and Manolis Koubarakis. Publish/Subscribe Functionalities for Future Digital Libraries using Structured Overlay Networks. In *Proceedings of the 8th International Workshop of the DELOS Network of Excellence on Digital Libraries on Future Digital Library Management Systems*, Schloss Dagstuhl, Germany, March 2005.

[202] Christos Tryfonopoulos, Stratos Idreos, and Manolis Koubarakis. LibraRing: An Architecture for Distributed Digital Libraries Based on DHTs. In *Proceedings of the 9th European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, pages 25–36, Vienna, Austria, September 2005.

[203] Christos Tryfonopoulos, Stratos Idreos, and Manolis Koubarakis. Publish/Subscribe Functionality in IR Environments using Structured Overlay Networks. In *Proceedings of the 28th Annual International ACM SIGIR Conference*, pages 322–329, Salvador, Brazil, August 2005.

[204] D. Tsoumakos and N. Roussopoulos. A Comparison of Peer-to-Peer Search Methods. In *Proceedings of the International Workshop on the Web and Databases (WebDB)*, pages 61–66, 2003.

[205] S. Voulgaris and M. van Steen. An Epidemic Protocol for Managing Routing Tables in Very Large Peer-to-Peer Networks. In *Proceedings of the International Workshop on Distributed Systems: Operations and Management (DSOM)*, Heidelberg, Germany, October 2003.

[206] S. Waterhouse. JXTA Search: Distributed Search for Distributed Networks. Technical report, Sun Microsystems, Inc., 2001.

[207] I.H. Witten, A. Moffat, and T.C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kauffman Publishing, San Francisco, 2nd edition, 1999.

[208] T.W. Yan and H. Garcia-Molina. The SIFT Information Dissemination System. *ACM Transactions on Database Systems (TODS)*, 1999.

[209] T.W. Yan and H. Garcia-Molina. Index Structures for Selective Dissemination of Information Under the Boolean Model. *ACM Transactions on Database Systems (TODS)*, 19(2):332–364, 1994.

[210] T.W. Yan and H. Garcia-Molina. Index Structures for Information Filtering under the Vector Space Model. *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 337–347, 1994.

[211] T.W. Yan and H. Garcia-Molina. The SIFT Information Dissemination System. *ACM Transactions on Database Systems (TODS)*, 24(4):529–565, 1999.

[212] B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. In *Proceedings of the International Conference on Data Engineering (ICDE)*, March 2003.

[213] B. Yang and H. Garcia-Molina. Improving Search in Peer-to-Peer Networks. In *Proceedings of the International Conference on Distributed Computing and Systems (ICDCS)*, pages 5–14, Vienna, Austria, July 2002. IEEE.

[214] B. Yang and H. Garcia-Molina. Comparing Hybrid Peer-to-Peer Systems. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 561–570, 2001.

[215] W.G. Yee and O. Frieder. On Search in Peer-to-Peer File Sharing Systems. In *The ACM Symposium for Applied Computing*, 2005.

[216] W.G. Yee and O. Frieder. The Design of PIRS, a Peer-to-Peer Information Retrieval System. In *Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, 2004.

[217] J.A. Yochum. A High-Speed Text Scanning Algorithm Utilising Least Frequent Trigraphs. In *IEEE Symposium on New Directions in Computing*, 1985.

[218] D. Zeinalipour-Yazti, V. Kalogeraki, and D. Gunopulos. On Constructing Internet-Scale P2P Information Retrieval Systems. In *Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, pages 136–150, 2004.

[219] D. Zeinalipour-Yazti, V. Kalogeraki, and D. Gunopulos. Exploiting locality for scalable information retrieval in peer-to-peer networks. *Information Systems*, 30(4):277–298, 2005.

[220] Y. Zhang and J. Callan. Maximum Likelihood Estimation for Filtering Thresholds. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 2001.