TOP-K RETRIEVAL IN PEER TO PEER NETWORKS

Von der Fakultät für Elektrotechnik und Informatik
der Universität Hannover
zur Erlangung des Grades eines

DOKTORS DER NATURWISSENSCHAFTEN

Dr. rer. nat.

genehmigte Dissertation von

Dipl.-Inform. Uwe Thaden

geboren am 16. Juni 1973 in Börger

2005

Referent: Prof. Dr. techn. Wolfgang Nejdl
Ko-Referent: Prof. Dr.-Ing. Christian Grimm
Tag der Promotion: 12. Dezember 2005

Im Lichte bereits erlangter Erkenntnis erscheint das glücklich Erreichte fast wie selbstverständlich, und jeder intelligente Student erfaßt es ohne zu große Mühe. Aber das ahnungsvolle, Jahre währende Suchen im Dunkeln mit seiner gespannten Sehnsucht, seiner Abwechslung von Zuversicht und Ermattung und seinem endlichen Durchbrechen zur Klarheit, das kennt nur, wer es selber erlebt hat.

Albert Einstein, 'Mein Weltbild', 1934

ACKNOWLEDGEMENTS

Scientific work and especially the creation of a PhD-thesis is not done isolated from other people. I want to thank my dissertation advisor Prof. Dr. techn. Wolfgang Nejdl for his support and for giving me the needed freedom to work the way I wanted. I wish to thank the co-referent Prof. Dr.-Ing. Christian Grimm for his time and comments! A *thank you* goes to my colleagues at KBS and L3S; just to name a few who were great partners in discussions and always willing to help: Wolf Siberski, Dr. Martin Wolpers, Dr. Wolf-Tilo Balke, Prof. Dr. Friedrich Steimann, Dr. Jörg Diederich, Fabian Leitritz, and Franziska Pfeffer and Katia Cappelli.

On the other hand I wish to thank my parents who always supported me in everything I did. And last but not least a big kiss to my wife Carola and to my daughter Claire Marie who gave me their love and help to encourage me every day — I love you!

ABSTRACT

Top-k Retrieval in Peer to Peer Networks

in

English

Distributed information systems have become a very important technology, both in research and industry. Especially peer to peer networks made their way from simple file sharing systems to matured systems in different contexts. When querying, users are interested in only a few results which match best for their query. From the information retrieval some techniques for ranked retrieval are well known. Most of them rely on information which is gathered from parts of or the whole document collection; this is called collection wide information. The first chapters of this thesis present the background from peer to peer and information retrieval; state of the art systems for peer to peer systems information retrieval systens are presented together with their individual drawbacks, leading to the main contribution of this thesis: A new algorithm ProToRaDo, which offers distributed ranking with respect to collection wide information. The algorithm will be discussed, mathematically proved and evaluated using an implemented simulation framework.

Keywords: Peer to Peer, Top-k, Information Retrieval

ABSTRACT

Top-k Retrieval in Peer to Peer Networks

in

Deutsch

Verteilte Informationssysteme, insbesondere Peer to Peer-Netzwerke, bekommen eine immer stärker werdende Bedeutung. Diese Arbeit stellt die Entwicklung von verteilten Retrieval- und Ranking-Methoden in Peer to Peer-Netzwerken dar. Ausgehend von der Grundlagendarstellung von Peer to Peer werden aktuelle Peer to Peer-Overlay Netzwerke diskutiert und im Kontext 'Information Retrieval' untersucht. Eine wesentliche Herausforderung für das Ranking in Peer to Peer-Netzwerken ist die Nutzung von bekannten Verfahren aus dem Information Retrieval, die Informationen erfordern, die aus großen Teilen oder der gesamten Menge von Informationsquellen hervorgehen. Solche Informationen werden collection wide information genannt. Bekannte Information Retrieval-Techniken werden im Peer to Peer-Kontext diskutiert; darauf aufbauend werden bestehende Lösungen aus dem Peer to Peer-Bereich dargestellt. Der Schwerpunkt meiner Arbeit wird somit in der Darstellung des entwickelten Algorithmus ProToRaDo für verteiltes Ranking unter Einbezug von collection wide information in Peer to Peer-Netzwerken liegen. ProToRaDo wird unter Peer to Peer- und Information Retrieval-Aspekten diskutiert, mathematisch belegt und mit Simulationsergebnissen evaluiert.

Schlagworte: Peer to Peer, Top-k, Information Retrieval

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . . **ii**

LIST OF FIGURES . . . . **viii**

LIST OF TABLES . . . . **x**

CHAPTER

I. Introduction 1

1.1 Motivation 2
1.2 Problem Statement and Research Challenges 3
1.3 Contribution of this Work 4
1.4 Thesis Outline 5

II. Peer to Peer Networks 7

2.1 Introduction 8
2.1.1 History 10
2.1.2 Non Functional Requirements 10
2.1.3 What is Peer to Peer? 11
2.1.3.1 Peer to Peer and Client / Server 13
2.1.3.2 Peer to Peer and Distributed Databases 14
2.1.3.3 Peer to Peer and Grids 15
2.1.4 Overlay Networks 16
2.2 Existing Systems/Approaches 17
2.2.1 Classification of Peer to Peer Networks 18
2.2.2 Basic Topologies and Routing 20
2.2.2.1 Unstructured Peer to Peer Networks 21
2.2.2.2 Structured, DHT Networks 25
2.2.2.3 Structured, Non DHT Networks 36

III. Search in Peer to Peer Networks 48

3.1 Background: Information Retrieval 49
3.1.1 History 51
3.1.2 Information Retrieval 53
3.1.3 Databases and Information Retrieval 53
3.1.4 Modeling 55
3.1.4.1 Building the Index 56
3.1.4.2 Working with the Index 59
3.1.4.3 Measurements / Evaluation 65

3.1.5 Summary and Conclusion . . . 66
3.2 Information Retrieval in Peer to Peer Networks . . . 67
3.2.1 Challenges . . . 67
3.2.2 Existing Systems / Approaches . . . 67
3.2.2.1 PlanetP . . . 68
3.2.2.2 Rumorama . . . 69
3.2.2.3 Minerva . . . 70
3.2.2.4 Adlib . . . 71
3.2.2.5 PeerSearch . . . 71
3.2.2.6 Pier . . . 72
3.2.2.7 PIRS . . . 73
3.2.2.8 Other Systems . . . 74
3.2.3 Open Problems . . . 76

IV. An Algorithm for Top-k Retrieval in Structured Peer to Peer Networks . 77

4.1 What is challenging? . . . 78
4.1.1 Example . . . 79
4.1.2 Concrete Challenges . . . 80
4.2 Ingredients . . . 81
4.2.1 Overview . . . 81
4.2.1.1 Storage, Distribution, and Computation . . . 81
4.2.1.2 Basic Strategy . . . 83
4.3 Algorithm Details . . . 84
4.3.1 Top-k Retrieval Algorithm . . . 84
4.3.2 Correctness and Optimality Results . . . 91
4.3.3 Extending to Content and Classification . . . 95
4.3.3.1 Example: News Corpora in Digital Libraries . . . 95
4.3.3.2 Query Processing . . . 95
4.3.4 IDF Index Entry Expiration . . . 98
4.3.5 Query Routing Indexes . . . 99

V. Evaluation . . . 100

5.1 Simulator Design . . . 101
5.1.1 Existing Simulators . . . 101
5.1.2 Simulator Implementation . . . 102
5.1.2.1 Query- and Resource Description . . . 103
5.1.2.2 Super Peer based Topology . . . 103
5.1.2.3 Connections (Network Characteristics) . . . 103
5.2 Evaluation: Content . . . 104
5.2.1 Data Setup . . . 104
5.2.1.1 Top-10 . . . 105
5.2.1.2 Top-1 . . . 105
5.2.1.3 Static scenario . . . 105
5.2.1.4 Dynamic scenario . . . 105
5.2.2 Hypotheses . . . 106
5.2.3 Results . . . 106
5.2.3.1 Static scenario . . . 106
5.2.3.2 Dynamic scenario . . . 107
5.2.4 Discussion . . . 109
5.2.4.1 Estimated vs. measured contacted peers . . . 109

5.3 Evaluation: Content and Classification . . . 110
5.3.1 Data Setup . . . 110
5.3.2 Results . . . 111
5.3.2.1 Index size . . . 111
5.3.2.2 Index Effectivity . . . 111

VI. Summary and Outlook . . . 114

6.1 Summary . . . 114
6.2 Outlook and Future Work . . . 115

APPENDICES . . . **119**
A Curriculum Vitæ . . . 120
B Publications . . . 121

BIBLIOGRAPHY . . . **123**

LIST OF FIGURES

Figure

1.1 Querying . . . 3

2.1 Overlay network . . . 17

2.2 Classification of peer to peer networks (the colors depict the generation aspect) . . 19

2.3 User interface of Napster client . . . 23

2.4 Querying Napster . . . 24

2.5 Flooding in Gnutella . . . 25

2.6 Distributed Hash Table; adapted from [83] . . . 26

2.7 Identifiers in a Chord ring . . . 27

2.8 Chord finger table . . . 28

2.9 Pastry: Routing from peer 37A0F1 with key B57B2D . . . 31

2.10 Zone splitting in CAN on node arrival . . . 32

2.11 Route from node 1 to a key with coordinate (x, y) in a 2-dimensional CAN topology 33

2.12 CAN as random tree . . . 33

2.13 A P-Grid tree. The arrow shows the mapping from a key to a peer. . . . 35

2.14 A super peer network . . . 39

2.15 HyperCuP . . . 41

2.16 Network topology construction . . . 42

2.17 Network topology construction continued . . . 44

2.18 Network topology construction continued . . . 45

2.19 Network topology construction continued . . . 46

2.20 A Hypercube and its spanningtree . . . 47

3.1 Information processing 50

3.2 A simple SQL query and the resulting table 55

3.3 Information retrieval system 56

3.4 Information retrieval process 57

3.5 Indexes 59

3.6 Routing index 59

3.7 Word frequency diagram (adapted from [84]) 61

3.8 Vector space model 64

4.1 Collection wide information 79

4.2 Ingredients of ProToRaDo 82

4.3 Querying a super peer based peer to peer network 90

5.1 Zipfian Query Distribution 104

5.2 Index Development (first 550 queries) 106

5.3 Index Development (10000 queries) 107

5.4 Contacted Peers per Query 108

5.5 Message per Query (top-1, static) 108

5.6 Contacted Peers per Query (top-1, dynamic) 109

5.7 Results 110

5.8 Index size 111

5.9 Coverage of query index 112

5.10 Coverage of category index 113

5.11 Contacted peers per query 113

6.1 Development of error rate over time 118

LIST OF TABLES

Table

2.1 A comparism of P2P and Server-based Model 14

2.2 Peer to peer systems; adapted from [83] 22

2.3 Pastry routing table for *nodeID* 37A0x 30

2.4 Example: Routing state of *nodeID* 37A0F1 31

2.5 Peer to peer systems; adapted from [83] 38

3.1 Book table 54

3.2 Publisher table 54

3.3 Author table 54

3.4 VSM weight computation 64

4.1 Term frequencies 80

4.2 Inverted document frequencies 80

4.3 TFxIDF values 80

CHAPTER I

Introduction

1.1 Motivation

One typical use of computers is to store information. Modern harddisks and other media allow to backup all data which is created in a person's life (images, photos, texts, etc.). Information can be represented structured as relation in a database, unstructured in textual document or as complex documents in a proprietary storage-system, e.g. systems for multimedia-objects in digital libraries. The search based on keyword queries is one of the mostly used approaches. Those queries contain at least one word which should describe what the user wants to retrieve from the computer system. If the systems stores non-textual information, mainly meta data are used to search for objects, e.g. author or title of an image.

It doesn't matter how much data is stored on a computer, an implication is that users also want to find and re-use data stored on their computers. Thus, sophisticated search methods and algorithms are needed, which allow users to search their data and extract the relevant information for specific needs.

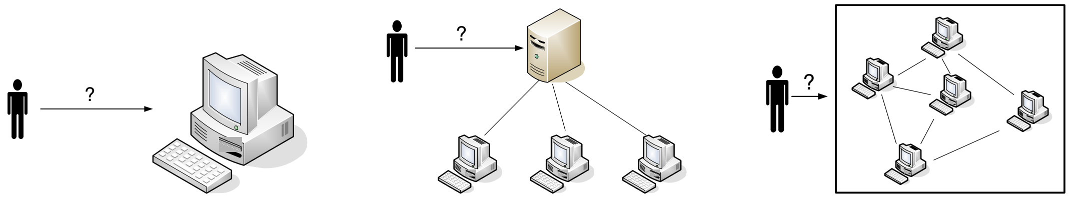
Users neither want to get an empty result set, nor do they want to get a large set of unsorted items which somehow match the search criteria. They want a limited list of only a few best matches. This best matches are named top-k, where k is the maximal number of items which should be listed in the result set. Retrieving the top-5 means, that the user will get a list of at most five items, where the first item matches best against the query. To evaluate this rankings, scoring or ranking functions are used. They measure how 'relevant' a result item is for the user, based on characteristics of the searched data.

To facilitate search and ranking researchers from the information retrieval community provide different approaches. The naïve way is to search all texts sequentially (even online). This is only appropriate when the amount of text is small (i.e., a few megabytes), but can be necessary if the text collection is modified very frequently.

Usually, the basic instrument for information retrieval is an index. The first well known indexes were used in books, where at the end an index lists all important terms together with the pages where they appear, allowing very fast and simple lookups of words.

An index is then used to find all relevant items (documents) for a result set. There are several methods to calculate how important (or relevant) a document is; most of them do not only use the information in the index, but also rely on some additional information which uses data that must be collected and/or aggregated from the whole set of documents. This is called *collection wide information.*

This is obviously necessary since the understanding of the term *importance* implies per se relationships among documents.



(a) Desktop (b) Server (c) Peer to peer

Figure 1.1: Querying

If all data is *stored locally on one computer* there is no problem in getting all information (including collection wide information); indexes can be build and maintained straightforward and ranking algorithms can be implemented. For the users this is all done transparently by presenting e.g. a desktop search engine. The user types in a query and the search engine can access and use all data which is directly available on the computer (see figure 1.1(a)).

Over the years computer were connected to build networks. Today it is common to pose queries not only to receive answers from the local machine, but queries are targeting networks of connected computers. The basic idea was to connect computers to exchange information, which implies that one must be able to query for information in such networks. To be able to reuse the highly developed information retrieval approaches, the *networks have one centralized point* where all needed information (index, etc.) is stored. A typical example for such centralized approaches are internet search engines like Google[24, 57], where of course the index is spread over many computers, but consequently this computer cluster (server farm) is treated as one big centralized point (see figure 1.1(b)).

In recent years, peer to peer networks[1] have gained much attention and proved to be an alternative to the client server model mentioned above. In a peer to peer network, all nodes are equal and independent of other nodes; peers can join and leave the network arbitrarily. Existing topologies and algorithms allow efficient lookup of data and routing to corresponding peers providing the information. The most important *difference to distributed systems is the missing centralized part.* There is no computer responsible for maintaining any information about the whole network (Figue 1.1(c)), which leads to several challenges regarding the query processing in such a decentralized system.

1.2 Problem Statement and Research Challenges

Peer to peer networks have become valuable infrastructures for sharing information in a wide variety of applications. The information shared can range from simple media files

[1] according to googlefight.com, in this thesis the notation *peer to peer* (87%) is used instead of *peer-to-peer* (13%)

annotated (and retrieved) with metadata (e.g. simple file sharing applications like Napster) to complex text or even multimedia documents, whose content has to be fully indexed. With such more complex documents recently also the dominant retrieval paradigms have shifted from simple exact matching of (metadata) attributes, also called query bindings, towards retrieving the most relevant objects in a ranked retrieval model.

From the information retrieval perspective a decentralization as done in peer to peer networks means a new challenge, since the mentioned data structures like indexes must be built and maintained completely decentralized, together with additional collection wide information, which allow to rank results coming from the peer to peer network using well know information retrieval techniques.

To approach this challenge, the areas of databases, information retrieval and peer to peer each offer a piece which put together solve the puzzle which leads into the direction of a solution.

Three main tasks can be defined as: Store and maintain information needed, find matching data in a distributed environment (i.e. query routing, show also hits that do not match exactly (ranking), and limit the number of entries in the results set (top-k.

1.3 Contribution of this Work

The main contribution of this work is ProToRaDo, which stands for Progressive Distributed Top-k Ranking of Documents.

ProToRaDo is a top-k answering and routing algorithm for peer to peer networks. It is the first approach in peer to peer which uses query driven updates of indexes maintaining collection wide information to allow for well known information retrieval methods for ranked retrieval of textual documents.

The algorithm addresses the problem of collecting and aggregating information in a peer to peer network without any instance that keeps any global information about the network.

This thesis offers solutions to the problems of designing indexing schemes for structured peer to peer environments with a view towards advanced retrieval paradigms. Highly efficient local indexing schemes that nevertheless deliver almost all relevant results are presented. Furthermore the problem of integrating (and updating) necessary collection-wide information in local query evaluation schemes, like e.g. inverted document frequencies, in local indexes is discussed.

For the evaluation of ProToRaDo a simulation framework was developed. It has open interfaces for e.g. plugging topologies or scoring methods. The simulation framework is written in Java and can be directly used for simulation of peer to peer networks having super peers and schema information. Changes for e.g. other topologies are straightforward and do not affect many part of the framework [108].

Besides the two concrete aspects mentioned this thesis also gives a very good example how traditional (databases, information retrieval) and modern (peer to peer) areas of computer science can be merged to achieve new results.

1.4 Thesis Outline

This thesis describes background information from peer to peer systems and information retrieval, discusses existing approaches from both areas, presents a new top-k algorithm and evaluates it in different settings.

In chapter II the main aspects of the peer to peer paradigm are described. Existing systems and approaches are discussed and relations to similar research topics are shown.

Chapter III introduces the needed background of the area of retrieval. The first sections give an overview of information retrieval methods, both in centralized and distributed systems. The second part part of the chapter discusses existing approaches using information retrieval techniques in peer to peer systems.

In chapter IV the new top-k-algorithm is presented. The concrete challenges are discussed. Afterwards the building blocks to solve the challenges are introduced, followed by a detailed discussion of ProToRaDo.

Chapter V introduces the simulation framework used in this work to evaluate the algorithm in different settings and scenarios. The chapter gives a detailed evaluation of ProToRaDo.

The conclusion is presented in chapter VI. Ongoing work and also ideas for next steps in the research direction of retrieval in peer to peer networks are briefly discussed.

This thesis is based on the author's work done at the 'Institut für Wissensbasierte Systeme' and the 'L3S Research Center' (both University of Hannover, supervisor Prof. Dr. Wolfgang Nejdl) in the area of peer to peer and information systems. A complete list of publications can be found in appendix B. The focus is on retrieval in peer to peer systems and discusses in detail the author's work, i.e.

1. *DL meets P2P - Distributed Document Retrieval*
   based on Classification and Content
   Co-authors: Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski
   ECDL, 2005 [21]

2. *Caching for Improved Retrieval in Peer-to-Peer Networks*
   Co-authors: Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski
   GI/ITG-Workshop, 2005 [20]

3. *Here is the News - Distributed Document Retrieval based on Classification and Content*
   Co-authors: Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski
   Technical Report, 2005

4. *Progressive Distributed Top k Retrieval in Peer-to-Peer Networks*
   Co-authors: Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski
   ICDE, 2005 [22]

5. *Top-k Query Evaluation for Schema-Based Peer-to-Peer Networks*
   Co-authors: Wolfgang Nejdl, Wolf Siberski, Wolf-Tilo Balke
   ISWC, 2004 [90]

6. *A Simulation framework for schema-based query routing in P2P-networks*
   Co-authors: Wolf Siberski
   P2P&DB, 2004 [108]

CHAPTER II

Peer to Peer Networks

Peer to peer computing didn't spring into existence in its current form, but has a history which led to the current ideas, approaches and challenges. This chapter gives an overview how peer to peer and the tight knit understanding evolved over time and discusses concrete approaches to peer to peer systems.

2.1 Introduction

Depending on the literature the term 'peer to peer' (P2P) is used alone or together with suffixes like 'system', 'network', 'architecture', 'technology', 'paradigm', or 'infrastructure'. Since it depends strongly on the context, in this thesis those suffixes are used interchangeable and if needed the specific context is shown and the appropriate term is chosen.

The term peer to peer became fashionable in the computing field around the end of the year 1999 when a system called 'Napster' allowed people to connect their computers directly for exchanging music files in MP3 format. Since then a lot of development took place today there is a wide range of ideas, approaches and systems in the context of the peer to peer paradigm.

A peer to peer network differs from conventional client server or multitiered server networks. A peer to peer architecture aims to provide direct communication between peers without the reliance on centralized servers or resources.

The claim for peer to peer architecture is that enables true distributed computing, creating networks of computing resources. Computers that have traditionally been used as clients can act as both clients and servers. Peer to peer allows systems to have temporary associations with one other for a while, and then separate. Peer to peer is an umbrella term for a rather diverse set of projects and systems including Napster, Freenet, and SETI@home.

The question is if peer to peer adds a new value. Gordon Moore postulated that the computing power doubles every 18–24 months. Thus, the power of desktop computers is at a very high level and still rapidly increasing, which enables users to participate in networks and actively bring in data and computer power (one of the main ideas of peer to peer). The value of this effect can be seen in Metcalfe's law, which states that 'The power of the network increases exponentially by the number of computers connected to it. Therefore, every computer added to the network both uses it as a resource while adding resources in a spiral of increasing value and choice.' – which is in fact peer to peer. Metcalfe's Law itself dates back to a slide that Bob Metcalfe created around 1980, when he was running 3Com, to sell the Ethernet standard. It was dubbed 'Metcalfe's Law' by George Gilder [59] in the 1990s.

The goals of peer to peer are:

- Cost sharing / reduction.
- Resource aggregation (improved performance).
- Improved scalability / reliability.
- Increased autonomy.
- Anonymity / privacy.
- Dynamism.
- Enabling ad-hoc communication and collaboration.

The main application of peer to peer is resource sharing, i.e.

- File sharing: In a group of peers each participant allows the other peers to access file locally stored. The (virtual) pool of files available for each peers is thus much larger than only its local files. Examples for this application: KaZaa, eMule.
- File storage: Storage capacities can be enlarged by using the peers to store (parts of) files. Companies can put large file split over the existing computers instead of buying expensive servers. An alternative is to use the existing computer to create replicas and backups.
- File archiving: An important issue for e.g. libraries and related archives is to ensure the availability of content for a long time. A distributed system ensures higher availability and increases storage capacity. Example: OceanStore [72].
- File distribution: Peer to peer system allow to use all bandwidth available from the participating computers to share files. Expensive servers and streaming costs can be compensated. Example: BitTorrent.
- Grid computing: Here, Grid computing can be seen as the aggregation of computing power, building e.g. virtual organizations (further details are in section 2.1.3.3).
- Communication: There are instant messengers that are built as peer to peer; another important thing is that the decentralized structure is more resistant against government censorship, which is still important in some countries[1]. Examples: ICQ, Guerilla Network Trading.

[1] A well known peer to peer system in that area is Peekabooty, http://www.peek-a-booty.org/

2.1.1 History

From an engineering perspective, the trend over the last decade, driven by forces such as enterprise application integration, has clearly been away from monolithic systems and toward distributed systems. This trend was inhibited somewhat by the ease of managing centralized applications, but the growth of the Internet, followed by the rise in importance of B2B transactions, made full-scale distributed computing a business necessity.

Intersecting this trend is the growth in the availability of powerful networked computers and inexpensive bandwidth. To be effective, peer to peer computing requires the availability of numerous, interconnected peers.

These two trends combined to form the perfect playground for peer to peer application research.

Nontechnical social issues were also important. Most of what's driving the current interest in peer to peer computing unarguably arose as a result of the popularity of products like Napster, Scour, Gnutella, and others of their ilk. They provided the 'killer applications' that put a subset of peer to peer technology in the hands of lots and lots of end users. That first-hand experience, in turn, raised awareness of the power of the peer to peer paradigm.

2.1.2 Non Functional Requirements

Studies like e.g. [17] have shown that $\sim$50% of the network traffic of a typical backbone is caused by peer to peer applications:

HTTP: 14.6 %	Edonkey: 37.5 %
FTP: 2.1 %	Kazaa: 7.8 %
NNTP: 1.9 %	Napster: 3.8 %
Other: 31.8 %	Gnutella: 0.3 %
$\sum$ Non-P2P: 50.4 %	$\sum$ P2P: 49.6 %

From the known values the peer to peer community projects increasing numbers concerning the traffic caused using peer to peer applications.

The growth of network usage due to peer to peer implies that there is a number of characteristics of a peer to peer system that are not related to the functionality, but are non functional requirements:

- Scalability (e.g. Connections, messages per request)
- Extendability (e.g. E.g. Incremental, exponential)
- Load balance (e.g. Fair distribution of responsibilities)

- Fault-tolerance (e.g. Handle membership dynamicity)
- Support for heterogeneity (e.g. Physical capabilities and behavior)
- Autonomy (e.g. Cooperation, self organization)
- Efficiency (e.g. Mapping to lower level networks, decreased maintenance)
- Security-related aspects (e.g. Resistant to attacks (e.g. DoS))

Obviously, to achieve the goals of peer to peer, one has to accept trade offs, which include:

1. Fault-tolerance vs. Heterogeneity, e.g. some topologies logarithmically increase connections for every peer
2. Scalability vs. Expandability, e.g. not all topologies are designed to extend incrementally
3. Heterogeneity vs. Load balance, e.g. in JXTA, rendezvous peers serve normal peers
4. Efficiency vs. Autonomy, e.g. in KaZaa, eDonkey, normal peers depend on super peers

2.1.3 What is Peer to Peer?

The previous sections introduce peer to peer in an informal way, giving examples how the idea evolved over the years. In the following, the common understanding of peer to peer is discussed based both on some popular definitions and the typical characteristics of a peer to peer network which distinguishes it of other known networks.

The term peer to peer is understood in many different ways; some of them are:

- Early and non-detailed attempts to describe peer to peer define peer to peer simply as the opposite of client/server architectures [110].
- Peer-to-Peer computing is a network-based computing model for applications where computers share resources via direct exchange between the participating computers. [23]
- Peer-to-Peer (P2P) refers to a class of systems and/or applications that use distributed resources in a decentralized and autonomous manner to achieve a goal e.g. perform a computation. [87]
- A distributed network architecture may be called a Peer-to-Peer (P-to-P, P2P, ...) network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers, ...). These shared resources are necessary to provide the Service and content offered by the network (e.g. file sharing or shared workspaces for collaboration). They are accessible by other peers

directly, without passing intermediary entities. The participants of such a network are thus resource (Service and content) providers as well as resource (Service and content) requestors (Servent-concept). [107]

- Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority. [16]
- P2P is a class of applications that takes advantage of resources – storage, cycles, content, human presence – available at the edges of the internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system, and have significant or total autonomy from central servers. That's it. That's what makes P2P distinctive. [127]
- Wray et al. [128] define Peer to peer networks just as a collection of heterogeneous distributed resources which are connected by a network.
- Peer-to-peer is a class of applications that take advantage of resources storage, cycles, content, human presence available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, peer-to-peer nodes must operate outside the DNS and have significant or total autonomy of central servers. [...] Sharing of computer resources by direct exchange. [95]

First and most important, peer to peer computing is the natural result of decentralizing trends in software engineering intersecting with available technology. Peer to peer means scalability and availability is not constraint by a single server (or a centralized group of servers), as experienced in the traditional client server architecture.

Basically, one can find three important characteristics of node in a peer to peer network:

1. Have an operational computer of server quality.
2. Have an addressing system that is independent of the DNS.
3. Able to cope with variable connectivity.

2.1.3.1 Peer to Peer and Client / Server

Castro et al. [36] states that there is no clear border between a client-server and a peer to peer model. Both models can be built on a spectrum of levels of characteristics (e.g., manageability, functionality (e.g., lookup versus discovery), organizations (e.g., hierarchy versus mesh), components (e.g., DNS), and protocols (e.g., IP), etc.

However, the most distinctive difference between client/server networking and peer to peer networking is the concept of an entity acting as a Servent, which is used in peer to peer networks. Servent is an artificial word which is derived from the first syllable of the term server ('Serv-') and the second syllable of the term client ('-ent'). Thus this term Servent shall represent the capability of the nodes of a peer to peer network of acting at the same time as server as well as a client.

Each node takes both role of client and server simultaneously, each node has equivalent capabilities and responsibilities. As a client, it can query and download its wanted objects from other nodes. As a server, it can provide objects to other nodes at the same time.

This is completely different to client/server networks, within which the participating nodes can either act as a server or act as a client but cannot embrace both capabilities [107].

The peer to peer model has a number of distinct advantages over the client/server model; such as: improving scalability by avoiding dependency on centralized points; eliminating the need for costly infrastructure by enabling direct communication among clients; and enabling resource aggregation [95]. The resources encompass computing power, data (storage and content), network bandwidth, and presence (computers, human and other resources); see table 2.1.

	Server-based model	**P2P model**
Information Ownership	Server-based, a single server or a pre-determine set of servers	Peer-based, dynamically adjusted, based on content availability
Content is organized	By domain	By peer group or topic
Content is backed-up On	Secondary or mirror site	Potentially, every peer can act as a mirror
Primary Endpoint behavior	As a client	As a client, a server, or a super-peer (rendezvous or router peer)
Information Retrieved	From the server	For closest edge peer
Address resolution & discovery mechanism for finding a host	Via name servers based on DNS entry; DHCP is possible with network address translation (NAT) service	Via the query response from the rendezvous peer or using peer discovery protocols
New content originates from	The central server and gets pushed to the edge	Any edge peer and propogates through out the network
Traffic Routing	Via router	Via router peer
Primary point of contact	A central server	A peer group, a topic, or a single peer
Communication among clients	Centrally coordinated by a central server	Self-regulated by peers themselves

Scalability	Limited by the server	Limited by the network
Where to add new content	On the server	On any peer
Effect of adding new clients	Increase resource demand / load on the server	Increase the number of available data sources
Fail-over / Interruption & Resume Service	Fail-over to backup site, when all backup sites are exhausted, the site is down	Fail-over to secondary peer
Message propagation Pattern	Radiate from the server, broadcasted from a central source	Propagates based on ad hoc connections between the peers, queries & responses
Denial of service (DOS) attack	Bring the main site down	Brings a single peer or peer group down
Bottleneck	The server or the network	The network
Cost-effective security implementation	On the server	On each peer, peer group, or superpeer
Content management & message filtering	On the server, centralized	Each peer makes its own decision based on query responses and comparative scores of multiple results

Table 2.1: A comparism of P2P and Server-based Model

2.1.3.2 Peer to Peer and Distributed Databases

There are several features that distinguish P2P systems from distributed database systems (DDBS); the main aspects are:

- In peer to systems, nodes can join and leave the network anytime. In DDBS, nodes are added to and removed from the network in a controlled manner, i.e., when there is a need for growth or retirement.
- Peer to peer systems usually do not have a predetermined (global) schema among nodes. Queries are largely based on lookups of key or keywords. In DDBS, nodes are typically stable and have some knowledge of a shared schema.
- In peer to peer systems, nodes may not contain the complete data. Further, nodes may not be connected. Thus, answers to queries are typically incomplete. In DDBS, one expects and can actually retrieve the complete set of answers.
- Data integrity is a very important aspect in databases. ACID stands for Atomicity, Consistency, Isolation, and Durability. They are considered to be the key transaction processing features of a database management system. Without them, the integrity of the database cannot be guaranteed. Peer to peer network can be very volatile; and thus transaction in the sense of database are not feasible. Brewer codifies this issues of ACID in his ‘CAP Conjecture’ [58], which states that a distributed data system can enjoy only two out of three of the following properties: Consistency, Availability, and tolerance of network Partitions. He notes that distributed databases always chose ‘C’, and sacrifice ‘A’ in the face of ‘P’.

2.1.3.3 Peer to Peer and Grids

The Grid (or simply Grids) is an emerging technology which is somehow' related to peer to peer, since both address the problem of organizing large scale computer networks. Thus, it's important to do both differentiate and merge the ideas of peer to peer and Grid.

Peer to peer networks and Grids are distributed computing models that enable decentralized collaboration.

Ian Foster is seen as the 'father' of Grid computing. In his papers he gives a very discussion what *Grid* means [52]. Regarding peer to peer and Grid Foster et al. [53] argues that

- both are concerned with the same general problem: organization of resource sharing within virtual communities;
- both take the same general approach of creating overlay structures that coexist with, but need not correspond in structure to, underlying organizational structures;
- each has made genuine technical advances, but each also has - in current instantiations - crucial limitations: 'Grid computing addresses infrastructure but not yet failure, whereas P2P addresses failure but not yet infrastructure'; and
- the complementary nature of the strengths and weaknesses of the two approaches suggests that the interests of the two communities are likely to grow closer over time.

Foster understands peer to peer in the common sense as e.g. [127] does, and defines Grids as:

'sharing environments implemented via the deployment of a persistent, standards-based service infrastructure that supports the creation of, and resource sharing within, distributed communities. Resources can be computers, storage space, sensors, software applications, and data, all connected through the Internet and a middleware software layer that provides basic services for security, monitoring, resource management, and so forth. Resources owned by various administrative organizations are shared under locally defined policies that specify what is shared, who is allowed to share, and under what conditions.' [53]

Compared to the peer to peer definitions given in section 2.1.3 the differences can be listed as:

- Target communities: Grid establishes homogenous communities with restricted participation, while peer to peer is open for anonymous individuals. A a consequence, in Grids trust is given, in peer to peer this is a hard task. In a peer to peer network, users can join and leave (anonymously) as they like.
- Peer to peer still focuses on exchange of files as resources. In Grids the shared re-

sources are more divers in type, e.g. files, storage, and computing power. This means peer to peer is less expensive compared to Grids in this sense.

- Grid applications are often complex and involve various combinations of data manipulation or computation, while peer to peer applications mainly focus on file sharing, number crunching, or content distribution.
- Compared regarding scale, Grids deal with a moderate number of entities, e.g. 10s of institutions and 1000s of users which use a larger amount of data in the sense of activity (>> 4TB/day). Peer to peer networks aim to scale up to millions of users, but the activity is less than in Grids.
- Finally, Grids aim to use standard protocols (Global Grid Forum, etc.) for a shared infrastructure (authentication, discovery, etc.). For peer to peer currently each application defines and deploys its own infrastructure.

As a summary the main difference between Grids and peer to peer is that the latter focuses more on scale and volatility while Grids aim to solve problems in the orthogonal direction of functionality and infrastructure.

2.1.4 Overlay Networks

A network defines addressing, routing, and service models for communication between hosts. Peer to peer networks build new connections, but not new physical topologies. They are virtual topologies, called overlay networks, based on some existing infrastructure like the internet: Overlay networks create a structured virtual topology above the basic transport protocol level that facilitates deterministic search and guarantees convergence. Thus, an overlay network adds an additional layer of indirection/virtualization and changes properties in one or more areas of underlying network.

Figure 4.1 illustrates this. In the lower part the physical network is shown; computers are connected and can exchange information via typical routing mechanisms like TCP/IP, which are mainly optimized for hop count. This means the services in the network are given, but limited by the physical infrastructure, e.g. in there physical protocol a discovery of some services is not possible, but an overlay network may establish a virtual possibility to locate them.

As the image 4.1 shows, there are particular connections in the physical layer; not all computers are connected (and the connections differ from the connections in the overlay network). This can have several reasons, e.g. routing constraints or barriers like firewalls. This implies that there are limitations, i.e. see each other in the network or find each other

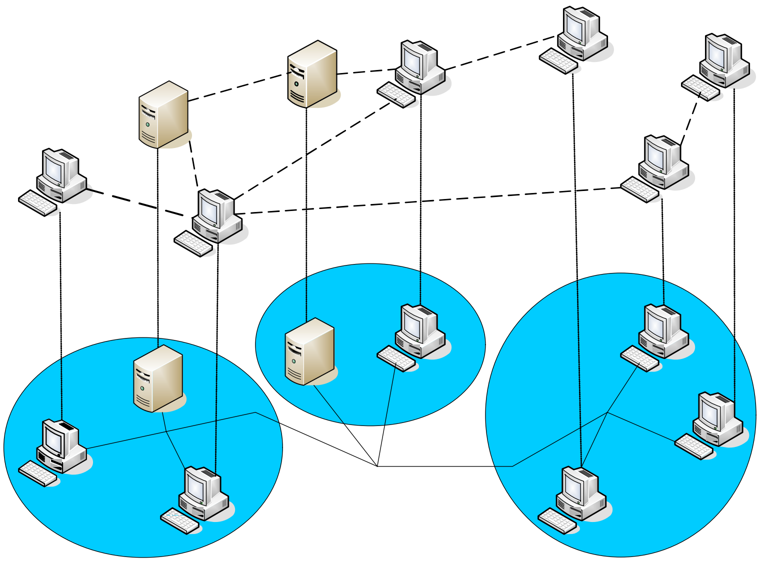


Figure 2.1: Overlay network

(which is nearly impossible with the physical layer based on addresses (uniform resource identifiers,URIs)).

For example, overlay networks can also be found in scenarios where IPv6-capable networks have to be connected over IPv4-only networks: In such scenarios, IPv6 packets are encapsulated into IPv4 packets at the edge between an IPv6/IPv4 networks and then sent over the IPv4 network according to IPv4 standard routing procedures. On arrival at the target IPv6 networks, the packets are unpacked and forwarded according to IPv6 standard routing to the final destination [6, 131].

2.2 Existing Systems/Approaches

This section introduces how peer to peer networks are organized.

Please note that in this section only peer to peer systems are discussed which are mainly concerned with routing in peer to peer networks. Systems which offer more sophisticated search options are discussed in section 3.2.

After introducing the basic concepts of overlay networks, the most important peer to peer approaches are discussed, closing with the HyperCuP, which is the topology used in the new ranking algorithm presented in chapter IV.

2.2.1 Classification of Peer to Peer Networks

The previous sections introduced the peer to peer paradigm by giving an historical background and the discrimination from related areas. After the first systems in the beginning of 2000 there has been much work on creating peer to peer networks; shortly after Napster had its enormous success, the drawbacks forced the development of improved routing mechanisms, which led to several approaches which are very different and sometimes even not directly comparable.

The current peer to peer networks are so many and each peer to peer network has so many different properties, that there is no unique classification criterion. In the following a presentation of a taxonomy is given which is widely accepted and used in this thesis.

The basic separation is based on the distinction whether a peer to peer network has an instance which maintains information about the whole network or not. Those networks (e.g. Napster) are called *centralized* and are not important for the upcoming discussion, since those both are not really peer to peer (i.e. central instance with global knowledge) and the have nearly the same characteristics as client server approaches.

Thus, in the following the *decentralized* peer to peer networks and their classification is introduced (see figure 2.2). In a decentralized peer to peer network all peers are understood to be equal; there is no peer which has information about the whole network.

Although the current peer to peer approaches can have different properties and also their goals can be different to some extent, two broad classes of infrastructure solutions have emerged, differentiated by the existence of global rules (or the lack of thereof) that may restrict the pattern of interactions between participating resources: unstructured and structured peer to peer systems for resource location.

In *unstructured* peer to peer systems in principle peers are unaware of the local storages that other (neighboring) peers in the overlay networks maintain. The search is done using flooding (forwarding the query over the network) queries to all neighboring peers all the time without acquiring any knowledge about the others peers' storages. As a consequence, they generate a large amount of messages per query which makes the approach poorly scalable in terms of communication cost when the number of peers grows. Furthermore, despite the large number of messages generated, there is no guarantee on the search success; especially it is difficult to find rare resources in an unstructured peer to peer network, since there is always a limitation in the flooding technique, mostly a maximal number of hops in the flooding process.

In contrast, in structured peer to peer systems peers maintain information about what resources other (neighboring) peers offer. Thus queries can be directed and in consequence substantially fewer messages are needed. This comes at the cost of increased maintenance efforts during changes in the overlay network as a result of peers joining or leaving. The most prominent class of approaches to structured P2P systems are distributed hash tables (DHT). They assign static identifiers to peers and the distributed data structures (e.g. DHTs) are constructed based on these identifiers by distributed algorithms.

The unstructured peer to peer networks can be further distinguished by the fact if they all peers are equal in the sense of authority or if some peers have locally (and sometimes dynamically) specific additional competencies. The latter type of networks are called *hybrid*, the first *pure* peer to peer networks.

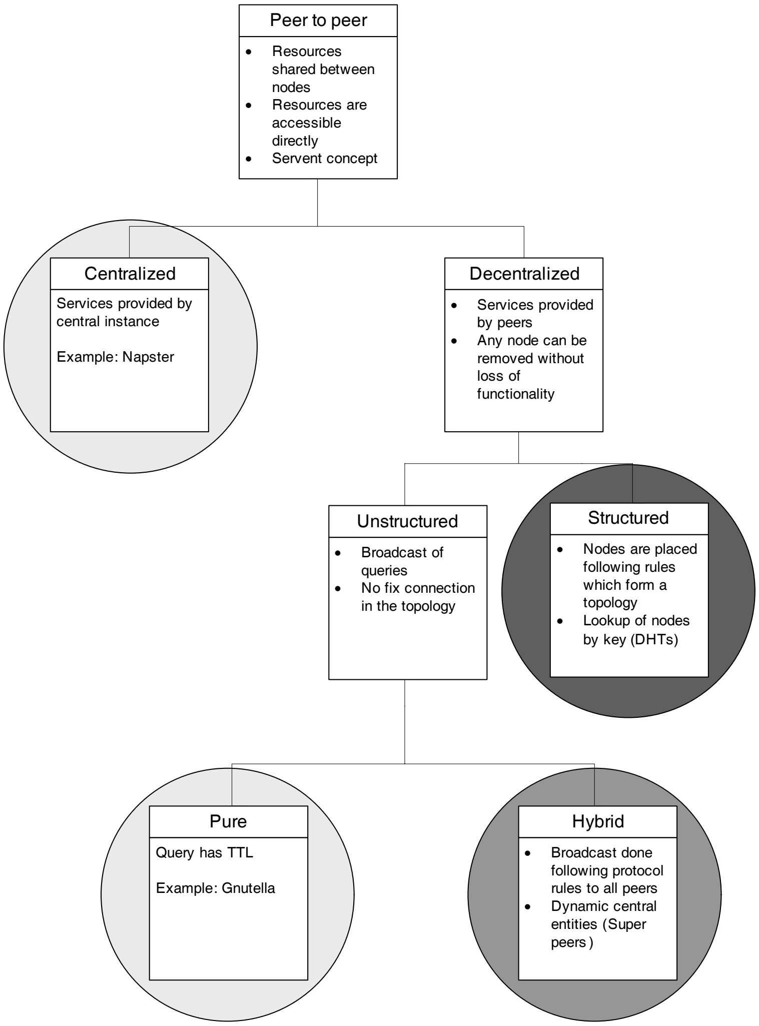

Peer to peer
Resources shared between nodes
Resources are accessible directly
Servent concept
Centralized
Services provided by central instance
Example: Napster
Decentralized
Services provided by peers
Any node can be removed without loss of functionality
Unstructured
Broadcast of queries
No fix connection in the topology
Structured
Nodes are placed following rules which form a topology
Lookup of nodes by key (DHTs)
Pure
Query has TTL
Example: Gnutella
Hybrid
Broadcast done following protocol rules to all peers
Dynamic central entities (Super peers)

Figure 2.2: Classification of peer to peer networks (the colors depict the generation aspect)

In a historical view the evolution is divided into *three generations*. The first generation covers the centralized and pure peer to peer networks; the second generation are the hybrid networks. The currently manifold discussed structured peer to peer network form the third generation (in figure 2.2 shown using different colors).

Tables 2.2 and 2.5 give an overview of current topologies, of which the most important are discussed in the following sections.

2.2.2 Basic Topologies and Routing

Peer to peer overlay networks allow to search for resources which are located at peers. The main task is to enable efficient search using sophisticated topologies. Daswani et al. [48] distinguish key based, keyword based and schema based systems.

The basic and simple concept in **key based search** is that resources are assigned an unique identifier. This is done using a hashing function, which is a specialized type of function used to convert data with a large range to a smaller (numerical) range, e.g. the filename (hash key) is hashed into a number (hash value), which can be looked up very fast. The drawback is that the hash value is the only attribute which can be searched for. Key based search is supported by all peer to peer networks based on distributed hash tables (DHT), cf. 2.2.2.2.1. Key based search does not allow for complex queries, but is a simple lookup.

Keyword based search is well known. Nearly every computer user has typed in some search terms in a search engine like Google [3]. System of this kind allow to search for documents based on a list of query terms; which can be combined with boolean operations such as ‘and’ (conjunctive query) or ‘or’ (disjunctive query).

Keyword search systems are divided by whether they provide ranking or not. Systems that do not rank the results present list of results where no information about the relevance of a hit is given; all hits in the result set are considered to be equal. The process which leads to such result lists is based on simple string or string pattern matching. Approaches that provide ranking have some methods to evaluate the results and assign each resource a score that relates it to other resources. Ranking/scoring can be done based on statistics derived from document full text (see section 3.1) or based on the link structure of resources [30].

The keyword based search can be done simply on attributes like e.g. the name of a music file. More challenging is the keyword search based on the content of the resources (mostly text documents). Keyword search based on the content of documents is one of the main challenges in information retrieval. Chapter III introduces the background from information retrieval, presents current approaches in the peer to peer area and motivates the importance of improving keyword based search in peer to peer systems.

Systems for **schema based search** manage and provide query capabilities for structured information. Structured means that the information adhere to predefined schemas. Modern systems use meta data to provide standardized schemas like Dublin Core[1] or IEEE Learning Object Metadata (LOM)[4]. A query is then described using schema attributes (or elements), e.g. if the schema has an element to list the author, a query could be: Find all documents with $author = `Miller'$. Schema based networks are not further discussed in this thesis; systems which use meta data are for example SWAM [67], Bibster [62], Edutella [91] and edusplash [2].

The next section present several topologies of peer to peer networks. Because of the different approaches, not all system are comparable; but where similarities are important, they are discussed.

2.2.2.1 Unstructured Peer to Peer Networks

Unstructured peer to peer networks are characterized by flooding queries through the network instead of controlled routing.

Flood search is the most primitive and an intuitive search mechanism, where a node broadcasts its search message to all its neighbors. If these neighbors do not posses the required resource, they in turn rebroadcast the search message to all their neighbors. Every broadcast is referred to as a 'hop' taken by the query request. The search mechanism sets a maximum number of hops, called as the time to live (TTL), after which the query is terminated.

2.2.2.1.1 Napster The first system which popularized the idea of peer to peer to million was Napster (Napster is used as both name of the company and the software client). Napster allowed[2] to locate and download music in MP3 format from a giant shared library of all its users collections in one convenient and easy to use interface. The only way to access other Napster user over the network was the client, which was simple and straightforward. No confusion for the end user. The system requirements to run the Napster client were very low: Users who wanted to search and share music files only needed a standard desktop PC and an internet connection. The features of the client were also very simple:

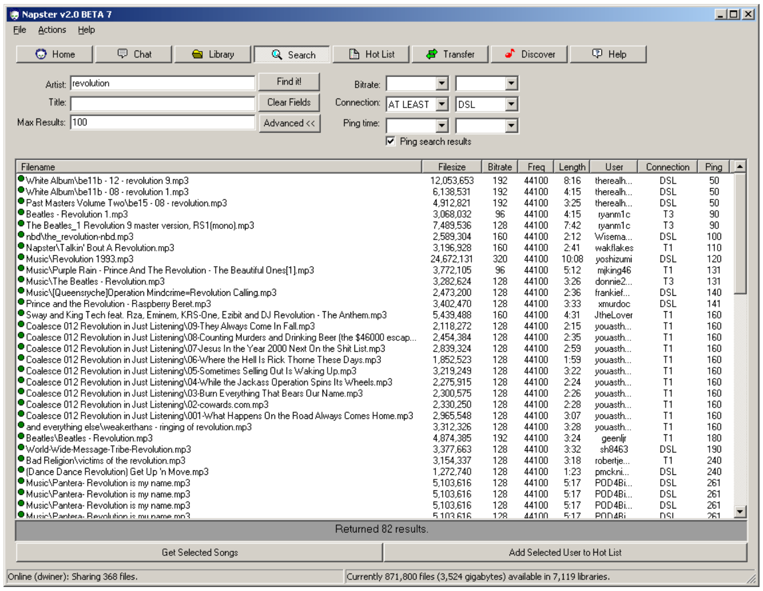
- Search - search for files based on title and artist and sort by bitrate, host connection, ping time and more
- Audio Player - plays MP3 files
- Hotlist - lets you keep track of your favorite MP3 libraries for later browsing

[2] The Napster for P2P sharing was closed in fall 2002 after some exciting court cases. Currently they sell software and latest news from June 2005 provide information about a collaboration with Ericsson to enable a mobile music platform.

Algorithm Taxonomy	P2P Overlay Network Comparisons				
	Freenet	Gnutella	FastTrack / KaZaA	BitTorrent	Overnet / eDonkey200
Decentralization	Loosely DHT functionality.	Topology is flat with equal peers.	No explicit central server. Peers are connected to their Super-Peers.	Cenralized model with a Tracker keeping track of peers.	Hybrid two-layer network composed of clients and servers.
Architecture	Keywords and descriptive text strings to identify data objects.	Flat and Ad-Hoc network of servents (peers). Flooding request and peers download directly.	Two-level hierarchical network of Super-Peers and peers.	Peers request information from a central Tracker.	Server provide the locations of files to requesting clients for download directly.
Lookup Protocol	Keys, Descriptive Text String search from peer to peer.	Query flooding.	Super-Peers.	Tracker.	Client-Server peers.
System Parameters	None.	None	None	*.torrent* file.	None
Routing Performance	Guarantee to locate data using *Key* search until the requests exceeded the Hops-To-Live (HTL) limits.	No guarantee to locate data; Improvements made in adapting Ultrapeer-Client topologies; Good performance for popular content.	Some degree of guarantee to locate data, since queries are routed to the Super-Peers which has a better scaling; Good performance for popular content.	Guarantee to locate data and guarantee performance for popular content.	Guarantee to locate data and guarantee performance for popular content.
Routing State	Constant	Constant	Constant	Constant but choking (temporary refusal to upload) may occur.	Constant
Peers Join / Leave	Constant	Constant	Constant	Constant	Constant with bootstrapping from other peer and connect to server to register files being shared.
Security	Low; Suffers from man-in-middle and Trojan attacks.	Low; Threats: flooding, malicious content, virus spreading, attack on queries, and denial of service attacks.	Low; Threads: flooding, malicious or fake content, viruses, etc. Spywares monitor the activities of peers in background.	Moderate; Centralized Tracker manage file transfer and allows more control which makes it much harder faking IP addresses, port numbers, etc.	Moderate; Similar threats as the FastTrack and BitTorrent.
Reliability / Fault Resiliency	No hierarchy or central point of failure exists.	Degradation of the performance; Peer receive multiple copies of replies from peers that have the data; Requester peers can retry.	The ordinary peers are reassigned to other Super-Peers.	The Tracker keeps track of the peers and availability of the pieces of files; Avoid Choking by fibrillation by changing the peer that is choked once every ten seconds.	Reconnecting to another server; Will not receive multiple replies from peers with available data.

Table 2.2: Peer to peer systems; adapted from [83]

- Chat - allows users to chat with each other in forums based on music genre
- Discover - find out about and download popular new releases and less-known artists



Artist: revolution
Title:
Max Results: 100
Bitrate:
Connection: AT LEAST DSL
Ping time:
☑ Ping search results

Filename	Filesize	Bitrate	Freq	Length	User	Connection	Ping
White Album\be11b - 12 - revolution 9.mp3	12,053,653	192	44100	8:16	therealh...	DSL	50
White Album\be11b - 08 - revolution 1.mp3	6,138,531	192	44100	4:15	therealh...	DSL	50
Past Masters Volume Two\be15 - 08 - revolution.mp3	4,912,821	192	44100	3:25	therealh...	DSL	50
Beatles - Revolution 1.mp3	3,068,032	96	44100	4:15	ryanm1c	T3	90
The Beatles_1 Revolution 9 master version, RS1(mono).mp3	7,489,536	128	44100	7:42	ryanm1c	T3	90
nbd\the_revolution-nbd.mp3	2,589,304	160	44100	2:12	Wisema...	DSL	100
Napster\Talkin' Bout A Revolution.mp3	3,196,928	160	44100	2:41	wakflakes	T1	110
Music\Revolution 1993.mp3	24,672,131	320	44100	10:08	yoshizumi	DSL	120
Music\Purple Rain - Prince And The Revolution - The Beautiful Ones[1].mp3	3,772,105	96	44100	5:12	mjking46	T1	131
Music\The Beatles - Revolution.mp3	3,282,624	128	44100	3:26	donnie2...	T3	131
Music\[Queensryche]Operation Mindcrime=Revolution Calling.mp3	2,473,200	128	44100	2:36	frankief...	DSL	140
Prince and the Revolution - Raspberry Beret.mp3	3,402,470	128	44100	3:33	xmurdoc	DSL	141
Sway and King Tech feat. Rza, Eminem, KRS-One, Ezibit and DJ Revolution - The Anthem.mp3	5,439,488	160	44100	4:31	JtheLover	T1	160
Coalesce 012 Revolution in Just Listening\09-They Always Come In Fall.mp3	2,118,272	128	44100	2:15	youasth...	T1	160
Coalesce 012 Revolution in Just Listening\08-Counting Murders and Drinking Beer (the $46000 escap...	2,454,384	128	44100	2:35	youasth...	T1	160
Coalesce 012 Revolution in Just Listening\07-Jesus In the Year 2000 Next On the Shit List.mp3	2,839,324	128	44100	2:59	youasth...	T1	160
Coalesce 012 Revolution in Just Listening\06-Where the Hell Is Rick Thorne These Days.mp3	1,852,523	128	44100	1:59	youasth...	T1	160
Coalesce 012 Revolution in Just Listening\05-Sometimes Selling Out Is Waking Up.mp3	3,219,249	128	44100	3:22	youasth...	T1	160
Coalesce 012 Revolution in Just Listening\04-While the Jackass Operation Spins Its Wheels.mp3	2,275,915	128	44100	2:24	youasth...	T1	160
Coalesce 012 Revolution in Just Listening\03-Burn Everything That Bears Our Name.mp3	2,300,575	128	44100	2:26	youasth...	T1	160
Coalesce 012 Revolution in Just Listening\02-cowards.com.mp3	2,330,250	128	44100	2:28	youasth...	T1	160
Coalesce 012 Revolution in Just Listening\001-What Happens On the Road Always Comes Home.mp3	2,965,548	128	44100	3:07	youasth...	T1	160
and everything else\weakerthans - ringing of revolution.mp3	3,312,326	128	44100	3:28	youasth...	T1	160
Beatles\Beatles - Revolution.mp3	4,874,385	192	44100	3:24	geenljr	T1	180
World-Wide-Message-Tribe-Revolution.mp3	3,377,663	128	44100	3:32	sh8463	DSL	190
Bad Religion\victims of the revolution.mp3	3,154,337	128	44100	3:18	robertje...	T1	240
(Dance Dance Revolution) Get Up 'n Move.mp3	1,272,740	128	44100	1:23	pmckni...	DSL	240
Music\Pantera- Revolution is my name.mp3	5,103,616	128	44100	5:17	POD4Bi...	DSL	261
Music\Pantera- Revolution is my name.mp3	5,103,616	128	44100	5:17	POD4Bi...	DSL	261
Music\Pantera- Revolution is my name.mp3	5,103,616	128	44100	5:17	POD4Bi...	DSL	261
Music\Pantera- Revolution is my name.mp3	5,103,616	128	44100	5:17	POD4Bi...	DSL	261

Returned 82 results.

Online (dwiner): Sharing 368 files.
Currently 871,800 files (3,524 gigabytes) available in 7,119 libraries.

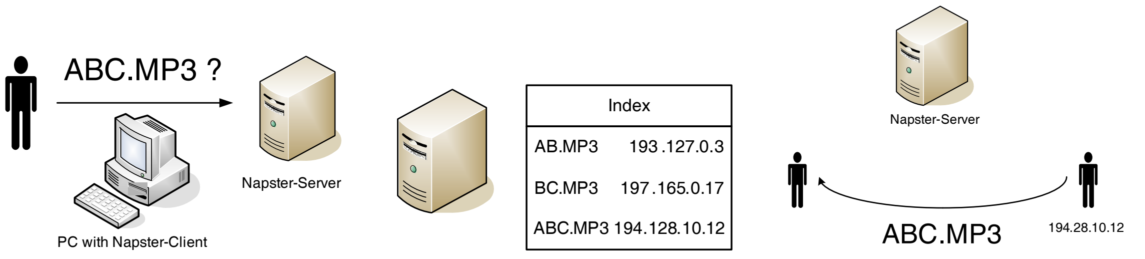
Figure 2.3: User interface of Napster client

The way Napster worked was very simple (figure 2.4): A central server had a list of all files that each Napster client (=user) had. If a user wanted a particular music file he did what was called 'to search Napster': The process of searching simply asked the central server: 'Does anyone have this file ?'. The central server looked etc. at its index of known files and provided the internet location as IP address of the other users who had the file. The download of the file was directly among the clients. Napster was introduced in mid 1999. End of 2000, 50 million users had downloaded Napster, in June 2001 the client has been downloaded by 65 million users, which made it the fastest growing application on the web [74].

The central server model made sense for many reasons – it was an efficient way to handle searches, and allowed Napster to retain control over the network. However, what it also meant was that when the lawyers came down on Napster, all they had to do was turn off the central servers and that was the end of Napster.

Thus, although hybrid systems like napster used a peer to peer communication model for the actual file transfer, the process of locating of the file is very much centralized. The two disadvantages that raises from that are:

1. Expensive: Since the names of all files from all users are stored in a central index on a server, the scalability is the bottleneck. Even if modern computer technology makes



Index	
AB.MP3	193 .127.0.3
BC.MP3	197.165.0.17
ABC.MP3	194.128.10.12

(a) User query (b) Server-lookup (c) Peer to peer transfer

Figure 2.4: Querying Napster

it possible to provide the needed computing power, it means immense administrative work.

2. Vulnerable: Even if server are clustered as a server farm, they consequently appear as one big computer and thus are mainly connected to the network as one monolithic system, which is then definitely one single point of failure. Here it can be ignored whether failure or attacks force the breakdown of such a system.

Nevertheless this first generation opened the area of peer to peer computing by facilitating bandwidth, digital content and social aspects of sharing (illegal intention).

2.2.2.1.2 Gnutella Gnutella was the second major peer to peer network that emerged. After Napster's demise, the creators of Gnutella wanted to create a decentralized network – one that could not be shut down by simply turning off a server. In the most basic sense, Gnutella worked by connecting users to other users directly (and bypassing any central server altogether). When a Gnutella client is started, it connects to a certain number of other users, and those users were connected to other users etc. in one giant network. Searching is done by flooding the query to connected neighbors, tagged with a time to live, i.e. 7 hops in Gnutella (see figure 2.5).

The main advantage was that it couldn't easily be shut down. The disadvantages were many – including slow searches and islands of sub-networks that weren't connected to each other.

2.2.2.1.3 FastTrack The drawbacks coming from the flooding in Gnutella inspired new system like e.g. FastTrack which is perhaps the most famous of this generation of networks. Note that systems like Kazaa, Grokster or Morpheus are all the same in connecting to the FastTrack network, they only differ in the client they offer to the users. FastTrack added a number of enhancements to the peer to peer networks, including supernodes, and

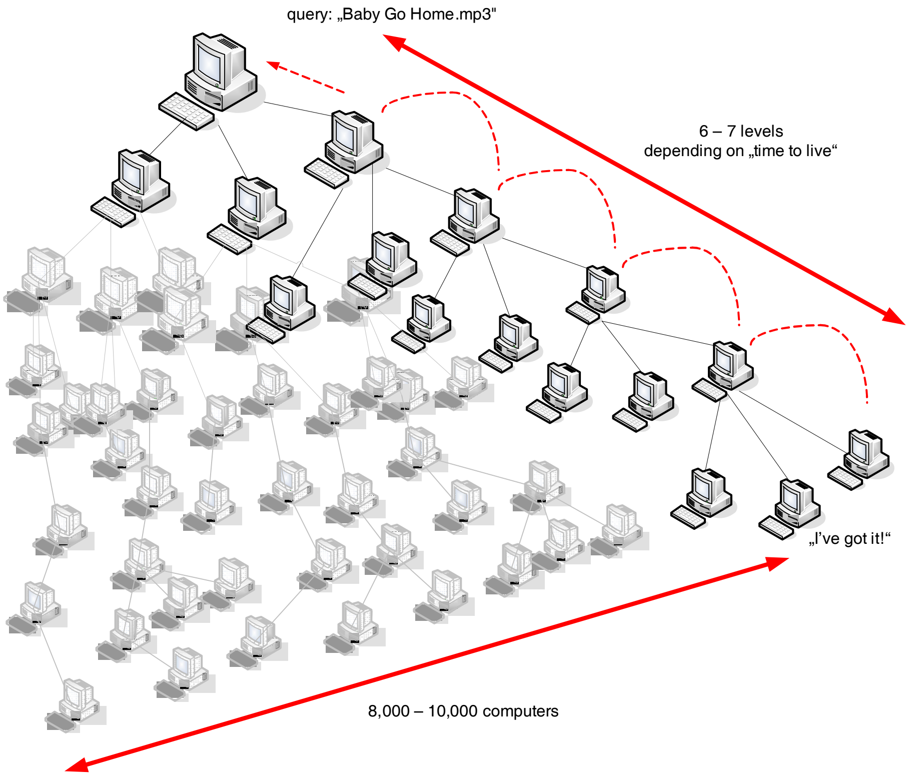


Figure 2.5: Flooding in Gnutella

spawning. These improvements both helped searches as well as download speeds.

2.2.2.2 Structured, DHT Networks

This section will discuss structured peer to peer networks based on distributed hash tables and briefly compare them. Over the years a lot of ideas were proposed and many of them are very similar; thus only the prominent approaches are presented, which proved to be utilizable over the years.

2.2.2.2.1 Distributed Hash Table (DHT) The task in a (structured) peer to peer network is to find the peer that has the information which is queries for. Structured peer to peer networks mainly use distributed hash tables. While hash tables are a well known data structure in the database community [97, 70, 76], the concept of distributed hash tables are introduced by peer to peer topologies.

In DHT systems, resources are associated with a key (produced, for instance, by hashing the file name) and each node in the system is responsible for storing a certain range of keys; i.e. the peers are 'hashed' into a position and are responsible for parts of the hash interval.

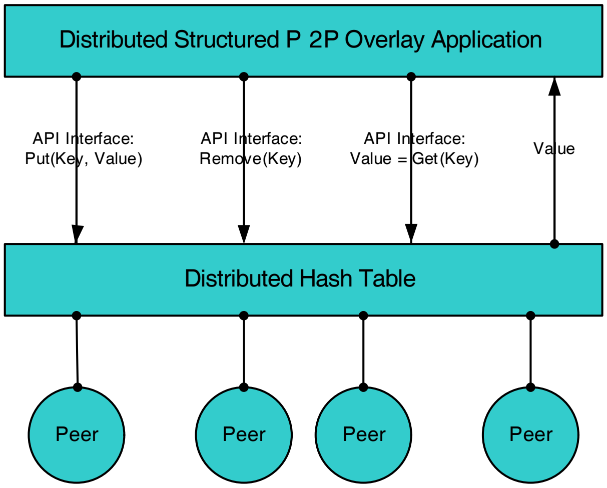

Distributed Structured P 2P Overlay Application
API Interface: Put(Key, Value)
API Interface: Remove(Key)
API Interface: Value = Get(Key)
Value
Distributed Hash Table
Peer
Peer
Peer
Peer

Figure 2.6: Distributed Hash Table; adapted from [83]

There is one basic operation in these DHT systems, lookup(key), which return the identity (e.g. the IP address) of the node storing the object with that key. This operation allows nodes to put and get resources based on the key, thereby supporting the hash table like interface, see figure 2.6. DHTs abstract from concrete data objects like documents, but offer a direct mapping from keys to values. A value can be an address, a document, or an arbitrary data item.

Although DHTs were initially motivated by the peer to peer file sharing systems, it was soon realized that the utility of DHTs in general is not limited to peer to peer systems and thus is already proving to be a useful substrate for large distributed systems. This includes distributed file systems [50, 72, 46], application layer multicast [100, 132], event notification [34, 102], and chat services.

2.2.2.2.2 Chord Chord, published by Stoica et al. in 2001, is one of the first topology protocols which uses a distributed hash table. It specifies how to find the locations of keys, how new nodes join the system, and how to recover from the failure (or planned departure) of existing nodes [113]. The protocol supports only one operation: Given a key, it maps the key onto a node.

Because of its simplicity, provable correctness and performance, Chord has become one of the most prominent protocols.

Identifier Space and Hashing The hash keys of the distributed hash table are m-bit identifiers, i.e. integer values in the range $[0, 2^m - 1]$. They form a one dimensional identifier circle modulo 2^m, which wraps around from $2^m - 1$ to 0. In Chord it is assumed that the identifier length m is chosen large enough to make the probability of two nodes or keys hashing to the same identifier negligible.

Chord uses consistent hashing, which has the specific tendency to to balance load, since each node receives roughly the same number of keys, and involves relatively little movement of keys when nodes join and leave the system.

Each node and each data item is assigned an identifier. The identifier of a node is calculated by hashing the node's IP address. A key identifier is produced by hashing the key. Since it is clear from the context, the term key is used for both the hash key and the corresponding hash value; the same applies to nodes.

A key k is assigned to the first node whose identifier is equal to or greater than k in the identifier space; The node is called the successor node of key k and is denoted by *successor(k)*. As mentioned above Chord can be thought as a circle modulo 2^m, which means that successor(k) is the first node clockwise from k and thus a node is responsible for all keys which precede it counter clockwise.

Example. Figure 2.7 shows an identifier ring with $m = 3$, i.e. $2^3 = 8$ identifiers. The circle has three nodes: 1, 2, and 5. The successor of identifier 1 is node 1, so key 1 would be located at node 1. Similarly, key 3 would be located at node 5. The circular modulo $2^3 = 8$ results in key 6 being located at node 2.

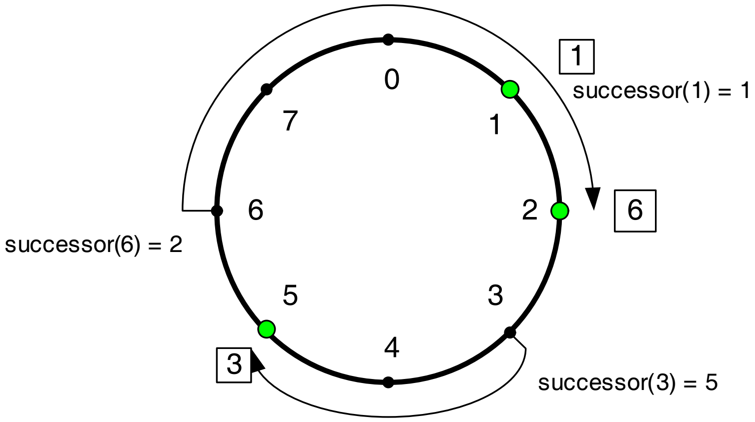
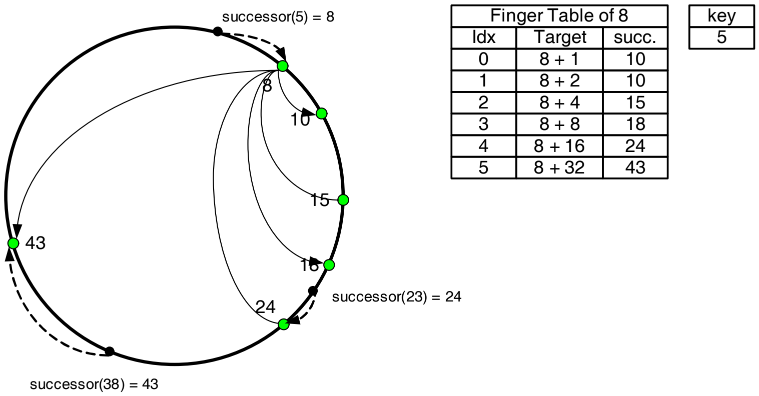


Figure 2.7: Identifiers in a Chord ring

Simple Key Location All identifiers are well ordered and keys and nodes are uniquely assigned (each key is hosted by a single, well defined node). The task is to quickly locate the node which is responsible for a particular key. The only information needed at each node is its successor node on the identifier circle. Each node can then forward a query to its successor. One of the nodes will determine that the searches key lies between itself and its successor. Consequently, the successor is then communicated as the result of the query back to its originator.

Finger Tables The simple key lookup using successor pointers only is not efficient and involves a number of messages linear to to the number of nodes on the identifier circle. Therefore, chord utilizes additional per-node state for more scalable key lookups. Each node n maintains a routing table with m entries, where m is 160 in the current Chord implementation using SHA-1 as hashing function. This routing table is called the finger table and each entry points to other nodes on the identifier circle.

The i^{th} entry in the table at node n contains the identity of the first node s that succeeds n by at least 2^{i-1} on the identifier circle, i.e. $s = successor(n + 2^{i-1})$, where $1 = i = 160$, and all arithmetic is modulo 2^{160}. The node s is called the i^{th} *finger* of node n. The first finger of n is its immediate successor on the circle.



Finger Table of 8		
Idx	Target	succ.
0	8 + 1	10
1	8 + 2	10
2	8 + 4	15
3	8 + 8	18
4	8 + 16	24
5	8 + 32	43

key
5

Figure 2.8: Chord finger table

Locating key using finger tables can intuitively be thought as cutting the search space in half in each iteration, see figure 2.8.

1. If successor(key) is present in finger table, then key is found.
2. If not, contact the last node in the finger table, that lies before the key. This node will then do the same until a node has been reached, that knows successor(key).

With high probability this requires O(log N) messages. The maximum number of hops in the worst case depends on the size of the finger table. The finger table contains a possible entry for each bit of the identifier length, i.e. 160 entries for the concrete Chord implementation based on SHA-1, which hashes into 160 bits. The number of peers is not relevant for the maximal hop count, e.g. if the network has only 20 peers there can be a route consisting of 19 hops when the key has a 'bad' position. Therefore, the log N guarantee is given with high probability, but not for every case.

Self Organization

When a peer joins the system, the successor pointers of some peers need to be changed. It is important that the successor pointers are up to date at any time because the correctness of lookups is not guaranteed otherwise. The Chord protocol uses a stabilization protocol [113] running periodically in the background to update the successor pointers and the entries in the finger table:

1. Initialize finger table and predecessor pointer.
   (a) The new node n asks existing node n^* to lookup the successor of n. This becomes n's successor and its predecessor becomes n's predecessor.
   (b) Then n asks n^* to lookup the keys in n's finger table.
2. Update finger tables, predecessor and successor pointers of existing nodes.
   (a) Runs through the identifier circle in reverse and updates the finger tables. n can calculate which nodes at which levels it might need to update.
   (b) Insert itself as the predecessor of its successor.
3. Move keys.

A peer can voluntarily leave the network of leave because of failure. The firstly mentioned case is simple to handle, because the leaving node can transfer its information (successor, keys, etc.). Afterwards all nodes which have the leaving node in their finger tables must be notified and the entries are replaced with the successor of the leaving node.

When peers fail, it is possible that a peer does not know its new successor. The solution in Chord is that each node does not only store its direct successor node, but a list of the x successor nodes in the identifier circle. This redundancy allow to compensate $x-1$ failures of nodes.

During the stabilization process nodes can determine if the direct successor is no longer available. A recovery algorithm then searches for a still existing node and asks it for its new successor. If a node n notes that its successor has changed, it sends a copy of its finger table to its next x neighbors.

2.2.2.2.3 Pastry Pastry from Rowstron et al. [101] is also one of the first DHT-systems. Tight-knit is PAST [50], which is a large scale, persistent peer to peer storage utility based on Pastry.

Identifier Space. Each node in the Pastry network is assigned a unique numerical 128-bit node identifier (*nodeId*), which indicates a node's position in a circular *nodeId* space. *NodeIds* are uniformly distributed by basing the *nodeId* on a cryptographic hash of the node's public key or IP address. Pastry views identifiers as strings of digits to the base 2^b, where b is typically chosen to be 4.

Given a message and a key, Pastry reliably routes the message to the node whose *nodeId* is numerically closest to the key. Instead of organizing the identifier space as a Chord like ring, the routing is based on numeric closeness of identifiers. In their work, Rowstron et al. [101] do not only focus on the number of routing hops, but also on network locality as factors in routing efficiency, which makes Pastry very interesting.

Routing Information As shown in tables 2.3, 2.4 and figure 2.9, each Pastry peer maintains a routing table, a neighborhood set, and a leaf set.

The routing table stores links to into the identifier space, which makes it comparable to Chord's finger table. On node n, the entries in row i store the identifiers of nodes whose IDs share an i-digit prefix with n, but differ in digit n itself. The routing table sorts *nodeIDs* by prefix; the first row of the routing table is populated with nodes that have no prefix in common with n (table 2.3).

The leaf set contains nodes which are close in the identifier space (like Chord's successor list).

Nodes that are close together in term of network locality are listed in the neighborhood set. In Pastry it is assumed the the network locality can be measured based on a given scalar network proximity metric, which is already available from the network infrastructure and might range from IP hops to actual geographical location of nodes.

0x	1x	2x	**3x**	4x	...	Dx	Ex	Fx
30x	31x	32x	...	**37x**	38x	...	3Ex	3Fx
370x	371x	372x	...	**37Ax**	37Bx	...	37Ex	37Fx
37A0x	37A1x	37A2x	...	37ABx	37ACx	37ADx	37AEx	37AFx

Table 2.3: Pastry routing table for *nodeID* 37A0x

If Pastry routes a message, at each routing step, a current node normally seeks to forward the message to a node with the *nodeId*, which shares a prefix with the key. The prefix is at least one digit (or b bits) longer than the prefix that the key shares with the current *nodeId*. If no such node is found in the routing table, the message is continuous to be forwarded to a node whose *nodeId* shares with the key the prefix. But the current node should be numerically closer to the key rather than the present node's id. Several such nodes normally can be found in the routing table; moreover, such a node must exist in the

NodeID 37A0F1			
Leaf Set (Smaller)			
37A001	37A011	37A022	37A033
37A044	37A055	37A066	37A077
Leaf Set (Larger)			
37A0F2	37A0F4	37A0F6	37A0F8
37A0FA	37A0FB	37A0FC	37A0FE
Neighborhood Set			
1A223B	1B3467	245AD0	2670AB
3612AB	37890A	390AF0	3912CD
46710A	477810	4881AB	490CDE
279DE0	290A0B	510A0C	5213EA
11345B	122167	16228A	19902D
221145	267221	28989C	199ABC

Table 2.4: Example: Routing state of *nodeID* 37A0F1

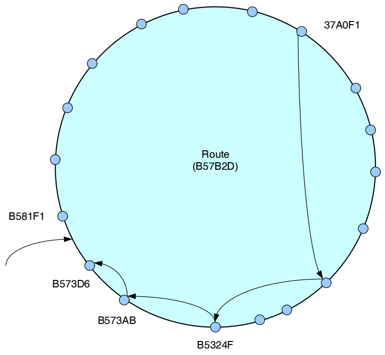


Figure 2.9: Pastry: Routing from peer 37A0F1 with key B57B2D

leaf set unless the message has already arrived at the node whose *nodeId* is numerically closest to the key or its immediate neighbor. The authors show that unless $\frac{l}{2}$ neighboring nodes in the leaf set have failed at the same time, at least one of those nodes must be live.

Node addition and failure. An important design issue in Pastry is how to efficiently and dynamically retain the node state, i.e. the routing table, leaf set and neighborhood sets, in the presence of new node arrivals, node failures, and node recoveries. When a new node with the newly chosen *nodeId* arrives, it can initialize its state tables, and then contact other nearby nodes of its presence, in terms to the proximity metric. Assuming the new node's *nodeId* is x and the nearby node is a, node x then asks a to route a special 'join' message using x as the key. Like any message, the join message is routed to the existing node z with *nodeId* numerically closest to x. In response to receiving the 'join' request, x can obtain the state tables from nodes a, z, and all nodes met on the path from a to z. The new node x inspects this information and then initializes its own state table with the probability of requesting state from additional nodes. Finally, x can notify any nodes that need to know its arrival. This procedure ensures that x is able to initialize its state using appropriate values and update the state in all other affected nodes [36, 101].

Nodes in the pastry network may fail or depart without indication. In Pastry network, that a node is considered failed means that its immediate neighbors cannot communicate with the node any longer. In order to handle node failures, neighboring nodes in the *nodeId* space should periodically exchange keep-alive message. If one node attempts to contact the failed node and there is unresponsive for a period, the failure in the routing table of the node is detected. Then all members of the failed node's leaf set update their leaf sets after receiving notification. Because of overlap of the nodes in the leaf sets with adjacent, this update is trivial. A node which is being recovered normally contacts the nodes in its last known leaf set and update its own leaf set by receiving their current leaf sets. Then it notifies the members of its new leaf set of its presence.

2.2.2.2.4 Content Addressable Network (CAN) In the same year in which Chord and Pastry were presented, Ratnasamy et al. introduced their work [99]. CAN is another distributed and structured peer to peer lookup services. Each key will be evenly hashed into a point of d-dimensional space as its identifier. When a node joins, it will randomly select a point of d-dimensional space. Then it will be responsible for half of regions this point belongs to, and hold all keys whose IDs belong to this region. For example, the first arrival node n_1 will be responsible for the whole space/region, the second arrival node n_2 will split whole region into two parts and takes one of them, and the third arrival node n_3 will split the n_1 region (if the random point that it selects belongs to this region) or n_2 region (otherwise) into two parts and takes one of them also (see figure 2.10).

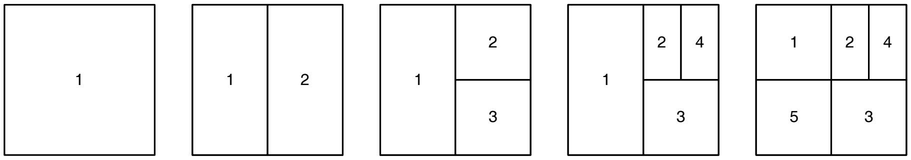


Figure 2.10: Zone splitting in CAN on node arrival

Routing. Intuitively, routing in CAN works by following the straight line path through the Cartesian space from source to destination coordinates. Each node will keep its neighbor node ID locally, and routing is then performed by forwarding requests to the regions closest to the position of the key, see figure 2.11.

Note that many different paths exist between two points in the space and so, even if one or more of a nodes neighbors were to crash, a node can automatically route along the next best available path

Maintenance via Random Trees The node management of CAN can be thought as random trees (figure 2.12). In a random tree new leafs are added randomly; if the chosen node is a parent, move down in the left or right part. Otherwise, the chosen node is a leaf;

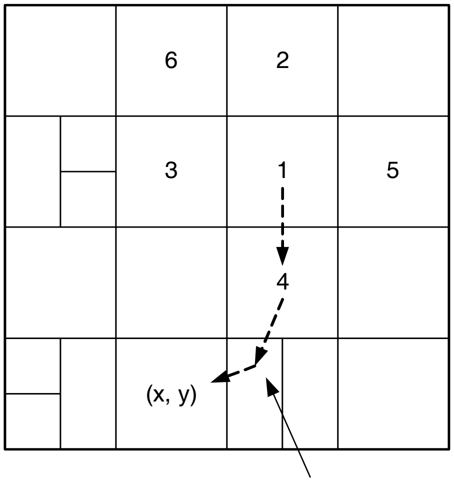


Figure 2.11: Route from node 1 to a key with coordinate (x, y) in a 2-dimensional CAN topology

then an additional leaf is added at the parent. The same holds for deleting nodes; when nodes leave a CAN, it must be ensured that the zones they occupied are taken over by the remaining nodes.

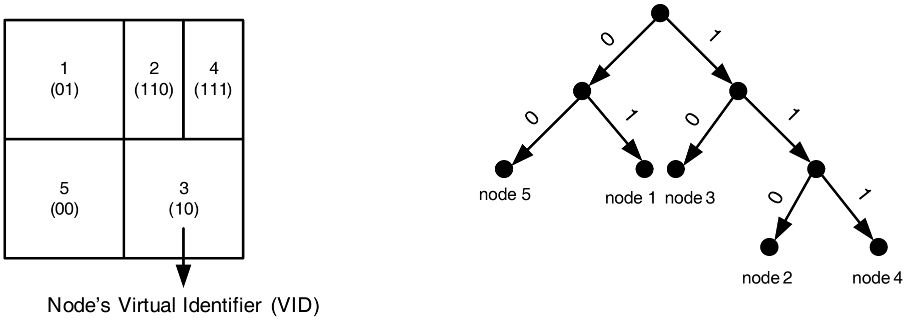


Figure 2.12: CAN as random tree

Nodes in the network test regularly if neighboring nodes have left the network. If so, the node that notices this takes over the CAN zone. Understanding CAN as a random tree makes it easy to introduce the aspect of **defragmentation**. Since frequent joining and leaving of nodes can lead to 'sectionalism' (fragmentation, where CAN zones are assigned peers randomly) a defragmentation method is needed. Such a zone reassignment works as follows: The normal procedure for doing the reassignment is for a node to explicitly hand over its zone and the associated (key, value) database to one of its neighbors. If the zone of one of the neighbors can be merged with the departing node's zone to produce a valid single zone, then this is done. If not, then the zone is handed to the neighbor whose current zone is smallest, and that node will then temporarily handle both zones. The CAN also needs to be robust to node or network failures, where one or more nodes simply become unreachable. This is handled through an immediate takeover algorithm that ensures one of the failed nodes neighbors takes over the zone. However in this case the (key,value) pairs held by the departing node are lost until the state is refreshed by the holders of the data.

The immediate takeover algorithm described above may result in a single node being assigned multiple zones (ideally, a a one-to-one assignment of nodes to zones should be retained, because this prevents the coordinate space from becoming highly fragmented). Therefore, a reassignment of this multiple assignment must be processed. Two cases for such a reassignment can be distinguished (taking the idea of random tree into account):

Neighboring zone is not divided If the sibling of this leaf is also a leaf (call it y) the hand-off is easy: simply coalesce leaves x and y, making their former parent vertex a leaf, and assign node y to that leaf. Thus zones x and y merge into a single zone which is assigned to node y.

Neighboring zone is divided If xs sibling y is not a leaf, perform a depth-first search in the subtree of the partition tree rooted at y until two sibling leaves are found. Call these leaves z and w. Combine z and w, making their former parent a leaf. Thus zones z and w are merged into a single zone, which is assigned to node z, and node w takes over zone x.

2.2.2.2.5 P-Grid P-Grid is a decentralized fully distributed search tree used for query routing [8, 9]. It uses binary keys to locate data items; it's a lookup system based on a virtual distributed search tree, similarly structured as standard distributed hash tables. Each node in the network can be thought as a possible route of a (typically binary) search tree (figure 2.13). Each level in the tree covers a smaller interval of nodes, until finally the bottom level a node holding the data object is found.

The peers in the P-Grid network use a few elements to guide searches. If peer p_i is a member of the P-Grid network it maintains the following:

- A k bit binary key $k_i := b_1 \dots b_k$, where k is less than or equal to n, for some bounded constant n which is the same for all p_i. This key denotes which subset of the key space that the peer is *responsible* for. Responsibility means that a p_i should be able to answer queries for keys that begin with the bit pattern k_i.
- An array (indexed from 1) with k elements, each containing a set of addresses to other peers. Index j in this array (j is less than or equal to k) contains the addresses of peers p_x that had the key $k_x := b_1 \dots b_{j-1}(1 - b_j)$ the last time peers p_i and p_x met. This array is called the *reference table* of the peer and the expression $REFS(l)$

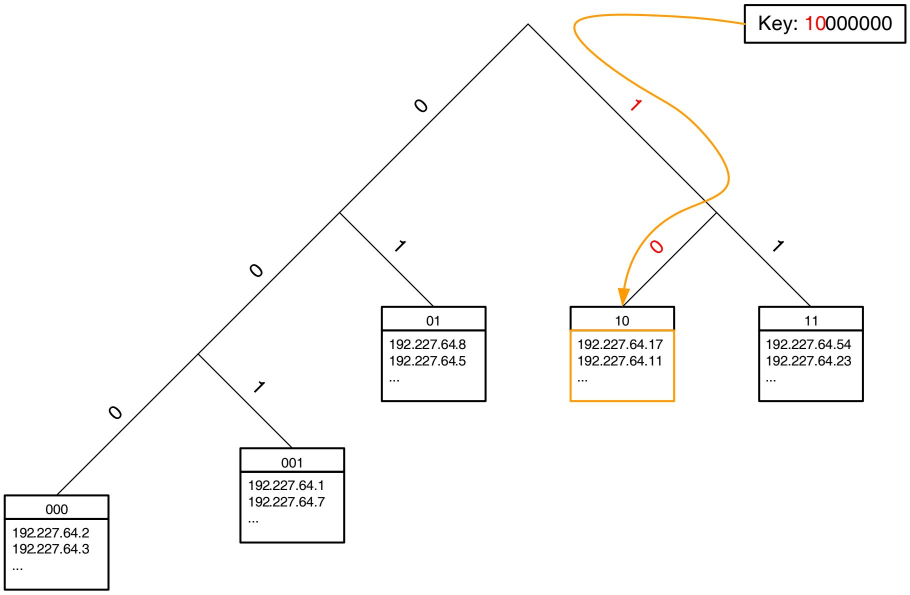


Figure 2.13: A P-Grid tree. The arrow shows the mapping from a key to a peer.

denotes the set of addresses at index l in the array. In P-Grid terminology this is called the *references at level* l.

- A set of data items D that match the key for the peer, and possibly an additional set of data items D^* that do not match. The second of these sets are for redundancy purposes, but is not strictly required. The descriptions of the P-Grid system actually vary on the description of data sets. Some descriptions does not even mention that the data set should contain elements matching the key.

Each peer in the P-Grid network is responsible for a specific key prefix. The length of this prefix grows as the peer communicates with other peers in the network.

An illustration of a P-Grid tree can be seen in figure 2.13. Each peer has a path, a binary string that gives the position of a peer in the tree. Peers can have the same path and become replicas of each other.

In order to distribute data items in a P-Grid network, items are mapped to binary strings, called keys. A peer is responsible for all data items with a key prefixed by the peers path. A simple mapping of keys to peers is illustrated in figure 2.13. The key 10000000 is mapped to the path 10. Every peer with this path is responsible of the key 10000000.

Searching. A peer performs a search for key k. To perform the search a connection to a peer p in the P-Grid network is established and the call $pgrid\_search(p, k, 0)$ is performed:

As shown, P-Grid differs from other approaches such as Chord, CAN, Pastry, etc. in

Algorithm 1 pgrid_search(p, k, i)

```
remain ← substring(p.key, i + 1, length(p.key));
common ← length(common_prefix(k, remain));
if p is responsible for this k then
  return p
end if
if someone else is responsible then
  k_next ← substring(k, common + 1, length(k))
  for each p_next in REFS(i + common + 1) do
    if online(p_next then
      host ← send_search(p_next, k_next, i + common);
      if host then
        return host
      end if
    end if
  end for
end if
return null
```

terms of practical applicability (especially in respect to dynamic network environments), algorithmic foundations (randomized algorithms with probabilistic guarantees), robustness, and flexibility. Similar to the bit length of the keys in Chord, the P-Grid protocol uses a maximal path length in the tree; otherwise the algorithm would not converge against a balanced tree. Furthermore, in P-Grid maintains replicas, which are responsible for the same key.

The **Update process** is divided in a push- and a pull-phase, which are executed separately, but maybe overlapping. In the push-phase a peer distributes the new version of a data element to some of its neighbors, which have the 'original' version and then forward the new version again to their neighbors, which is a kind of flooding. The pull-phase is initiated by a peer which enters the network again or which did not get any update for a longer period of time to ensure getting the current updates.

2.2.2.3 Structured, Non DHT Networks

As shown distributed hash tables are very well suited for exact matches, i.e. key lookups. Although there is some work on improving DHTs for allowing complex queries, there are still drawbacks with DHTs.

Especially for information retrieval, which focuses on the content based search in combination with ‘relevance’ of results, there is no adequate solution available (for a detailed discussion see section 3.1).

- All new content in the network has to be published at the node for the respective key, if new data on a peers arrives or a new peer joins the network. And in the case that a peers leaves, all data must be unpublished. Moreover, if a new document is added to any peer’s collection it will usually contain a large set of various terms that need to be indexed.
- Since in DHTs a hashing function decides on what peer the index for each term resides, chances are that a considerable number of peers holding some part of the DHT have to be addressed to fully publish all the information about the new document [56].
- Loo et al. show in [77] that due to the publishing / unpublishing overhead, distributed hash tables lack efficiency when highly replicated items are requested. In practical settings, the authors proved DHTs to perform even worse than flooding approaches; degrading even further, if stronger network churn is introduced.
- Another problem with distributed hash tables is that the retrieval uses exact matches of single keys, whereas information retrieval queries are usually conjunctions of several keywords. If such a query has to be answered using DHTs the peers offering content for each of the keywords have to be retrieved [68]. The intersection of the individual peer lists then may offer documents also relevant to the conjunctive query. However, there is still no guarantee that a peer in this intersection offers relevant content, because the publishing of he peer for each keyword may have been based on different documents.

Solutions to the mentioned aspects are still very premature, see e.g. [123, 78].

Thus, the flooding approach still exists, but has been improved over time. In the following the most important idea of enhancing flooding is introduced: Super peers which form a hybrid network in that sense that they each are responsible for some part of the network, but none of the super peers has neither a global administration task nor any information about the whole network.

The goal is a minimization of messages during a broadcast, that reaches all nodes in the network, not only a limited number of peers, e.g. all nodes that can be reached within a time to live of 7 hop counts.

2.2.2.3.1 Super Peers In typical real world peer to peer networks peers often considerably vary in bandwidth and computing power. Exploiting these different capabilities

Algorithm	P2P Overlay Network Comparisons					
Taxonomy	CAN	Chord	Tapestry	Pastry	Kademlia	Viceroy
Decentralization	DHT funcionality on Internet-like scale					
Architecture	Multi-dimensional ID coordinate space.	Uni-directional and Circular NodeID space.	Plaxton-style global mesh network.	Plaxton-style global mesh network.	XOR metric for distance between points in the key space.	Butterfly network with connected ring of predecessor and successor links, data managed by servers.
Lookup Protocol	key, value pairs to map a point P in the coordinate space using uniform hash function.	Matching Key and NodeID.	Matching suffix in NodeID.	Matching Key and prefix in NodeID.	Matching Key and NodeID based routing.	Routing through levels of tree until a peer is reached with no downlinks; vicinity search performed using ring and levelring links.
System Parameters	N-number of peers in network d-number of dimensions.	N-number of peers in network.	N-number of peers in network B-base of the choosen peer identifier.	N-number of peers in network b-number of bits ($B = 2^b$) used for the base of the choosen identifier.	N-number of peers in network b-number of bits ($B = 2^b$) of NodeID.	N-number of peers in network.
Routing Performance	$O(d \cdot N^{\frac{1}{d}})$	$O(\log N)$	$O(\log_B N)$	$O(\log_B N)$	$O(\log_B N) + c$ where c, = small constant	$O(\log N)$
Routing State	$2d$	$\log N$	$\log_B N$	$B \cdot \log_B N + B \cdot \log_B N$	$B \cdot \log_B N + B$	$\log N$
Peers Join / Leave	$2d$	$(\log N)^2$	$\log_B N$	$\log_B N$	$\log_B N + c$, where c = small constant	$\log N$
Security	Low level. Suffers from main-in-middle and Trojan attacks					
Reliability / Fault Resiliency	Failure of peers will not cause networkwide failure. Multiple peers responsible for each data item. On failures, application retries.	Failure of peers will not cause networkwide failure. Replicate data on multiple consecutive peers. On failures, application retries.	Failure of peers will not cause networkwide failure. Replicate data across multiple peers. Keep track of multiple paths to each peer.	Failure of peers will not cause networkwide failure. Replicate data across multiple peers. Keep track of multiple paths to each peer.	Failure of peers will not cause networkwide failure. Replicate data across multiple peers.	Failure of peers will not cause networkwide failure. Load incurred by lookups routing evenly ditributed among participating lookup servers.

Table 2.5: Peer to peer systems; adapted from [83]

leads to an efficient network architecture, where a small subset of peers, called super peers, takes over specific responsibilities and thus create a structure for the network.

Super peer networks occupy the middle ground between centralized and entirely symmetric peer to peer networks. They introduce hierarchy into the network in the form of super peer nodes, peers which have extra capabilities and duties in the network.

Super peer based networks can provide better scalability than unstructured networks, and are able to support sophisticated indexing and routing [129]. Queries can then be forwarded efficiently through a high bandwidth super peer backbone. Though usually also low bandwidth peers will be connected to that backbone, these peers are never used for forwarding queries or index information, but can fully dedicate their limited capabilities to answer queries.

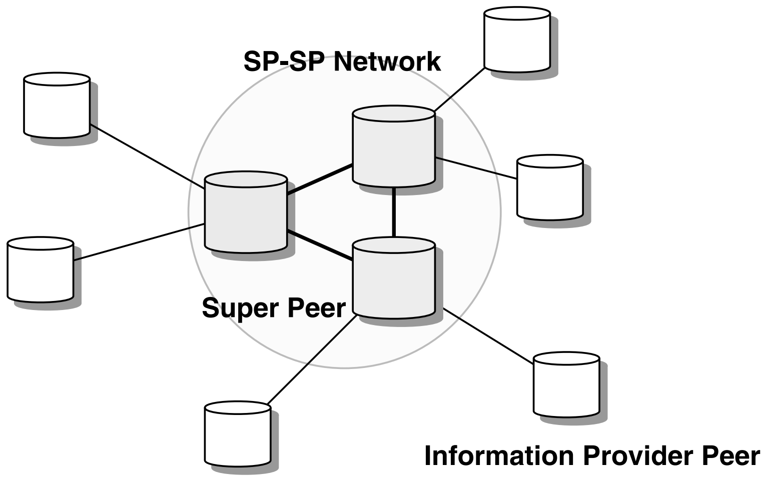


Figure 2.14: A super peer network

Thus, a super peer is a node that acts as a centralized server to a subset of clients, e.g. information provider and information consumer. Clients submit queries to their super peer node and receive results from it, as in a hybrid system. However, super peers are also connected to each other as peers in a pure system are (see also figure 2.14), routing messages over this overlay network, and submitting and answering queries on behalf of their clients and themselves. Examples of super peer networks are JXTA[10], Edutella [91] or Morpheus, KaZaa. Because a super peer network combines elements of both pure and hybrid systems, it has the potential to combine the efficiency of a centralized search with the autonomy, load balancing [129], robustness to attacks and at least semantic interoperability [10] provided by distributed search [80].

Research by Yang et al. [129] presents a design strategy for super peer networks. They present a research on unstructured (Gnutella like) networks and also present a detailed analysis of arguments in designing such systems. They claim that super peer networks are used to strike a balance between the inherent efficiency of centralized search, and the autonomy, load balancing and robustness to attacks provided by distributed search. Also that the amount of heterogeneity of capabilities (e.g. bandwidth, processing power) across peers can be used to advantage of the network.

In the unstructured networks, a super peer is a node that acts as a centralized server to a subset of clients. As in a hybrid system, clients/nodes submit queries to their super peer and receive results from it. However, super peers are also connected to each other as peers in a pure system are, routing messages over this overlay network, and submitting and answering queries on behalf of their clients and themselves. Thus, super peers are equal in terms of search, and all peers (including clients) are equal in terms of download. A ‘super peer network’ is simply a peer to peer network consisting of these super peers and their connected client nodes.

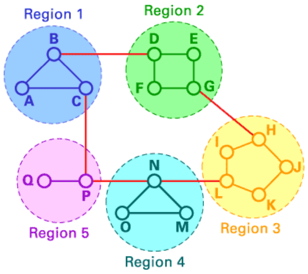
Super peer network combines the elements of both pure and hybrid systems. It has the potential to combine the efficiency of a centralized search with the autonomy, load balancing and robustness to attacks provided by distributed search. Queries can be handled quite efficiently and since there are so may super peer nodes no single super peer is burdened.

Super Peers in Other Contexts The idea of hierarchical structures like super peers is also known and used in other contexts, of which two are briefly described in the following.

Hierarchical routing:

The complex problem of routing in large networks can be improved by splitting such a network in a hierarchy of smaller networks (also called regions). Each of those networks is then responsible for its routing (see figure on the right) [118].

Example: The internet has three hierarchy levels: the backbones, the mid-levels and the autonomous systems. The backbones know how to route between the mid-level-networks, which know how to connect and route between the autonomous systems. Finally, the autonomous systems route internally.



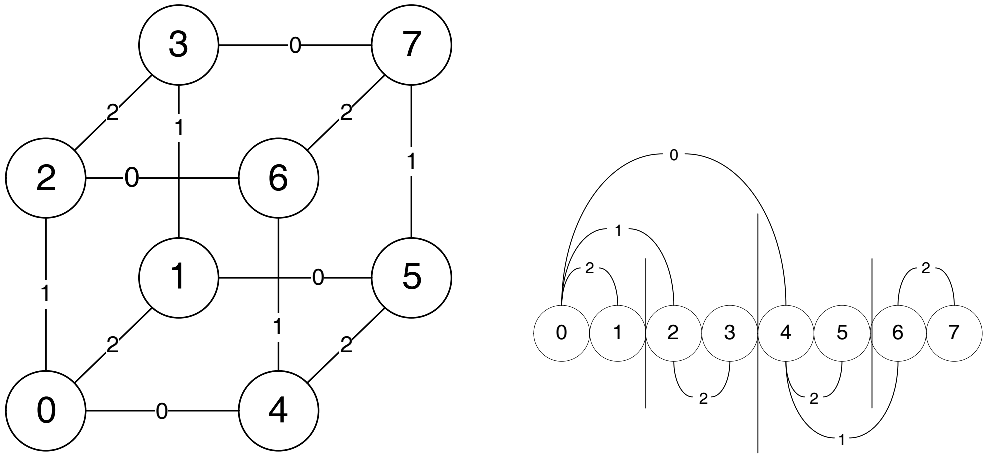
Sensor networks: In the area of sensor networks, hierarchies are used to aggregate information. On each level in the hierarchy, an aggregation of the data coming from the connection sensors is done and provided to the next level in the hierarchy. Another use of such hierarchies is to introduce separate protocols, e.g. for sensing, communications, and command in distributed sensor networks [31].

2.2.2.3.2 HyperCuP This section introduces the HyperCuP topology [105], which was developed at KBS and L3S. Analytical results are discussed in [98]. The following section is adapted from [105].

The HyperCuP algorithm is capable of organizing peers into a recursive graph structure from the family of Cayley graphs, out of which the hypercube is the most well known topology. This topology allows for $\log_2 N$ path length and $\log_2 N$ number of neighbors, where N is the total number of nodes in the network.

Broadcast and Search Algorithm The broadcast of the HyperCuP scheme guarantees that the set of nodes traversed strictly increases during a forwarding process, i.e. nodes receive a message exactly once. It is guaranteed that exactly $N - 1$ messages are required to reach all nodes in a topology. Furthermore, the last nodes are reached after $\log_b N$ forwarding steps. Any node can be the origin of a broadcast in the network, satisfying a crucial requirement (i.e. each node can be the root of a spanning tree).

The algorithm works as follows: A node invoking a broadcast sends the broadcast message to all its neighbors, tagging it with the edge label on which the message was sent. Nodes receiving the message restrict the forwarding of the message to those links tagged with higher edge labels.



(a) Hypercube graph

(b) Serialized notation

Figure 2.15: HyperCuP

As an example, refer to the serialized notation of the network graph in figure 2.15(b) (for clarity, only the links used in the example are depicted - however, one can just copy all links in figure 1a into this notation to arrive at the full picture): Node 0 sends a broadcast - at first to all its own neighbors, viz. nodes 4, 2 and 1. Node 4 receives the message on a link tagged as a level 0 link, i.e. it forwards the message only to its 1- and 2-neighbors, namely 6 and 5. At the same time, node 2 which has received the message on a level 1 link forwards it to its 2-neighbor, node 3. In the third forwarding step, node 6 relays the message to node 7, again its 3-neighbor.

Building and Maintaining Hypercube Graphs In the following, a distributed algorithm which allows nodes to build a hypercube topology is outlined. Here, the major challenges in peer to peer networks are as follows: To maintain network symmetry, any node in the network should be allowed to accept and integrate new nodes into the network. Furthermore, joining and leaving the network are to consume a reasonable amount of message transmission to limit the traffic imposed on the transport network. Clearly, a joining node should not have to register with all nodes in the network.

Topology Construction and Maintenance Algorithm In the following, the construction and maintenance of a hypercube peer to peer topology is described. The formal description of the algorithm and a proof of its completeness can be found in [106]. The algorithm is explained with an example scenario, having 9 peers joining a network, and one peer leaving during the process. The construction and maintenance algorithm is based on the notion that nodes in an evolving hypercube graph take over responsibility for more than one position in the hypercube. The idea is to have the hypercube topology of the next biggest complete hypercube graph already implicitly present in the current topology state, i.e. in the sets of all participating nodes. Upon arrival of new nodes, the complete hypercube topology unfolds as needed. Upon removal of nodes, other nodes jump in to cover the positions previously covered by the node that left the topology, prepared to give these positions up again as new nodes join. Since the complete hypercube topology is implicitly preserved, the broadcast and search algorithms do not have to change either - still, every peer receives a broadcast message exactly once.

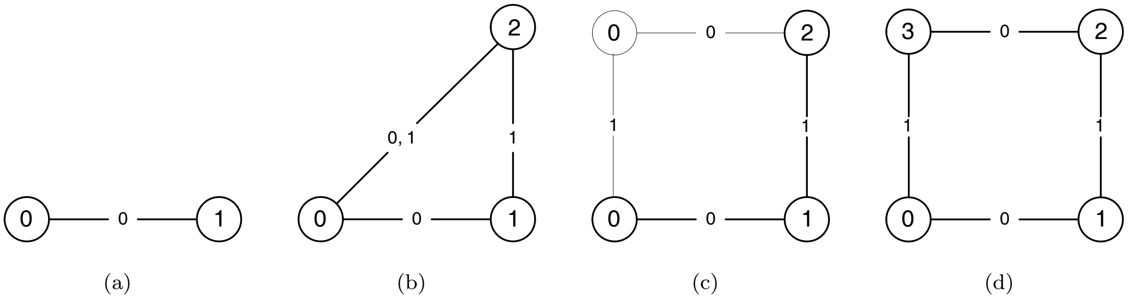


Figure 2.16: Network topology construction

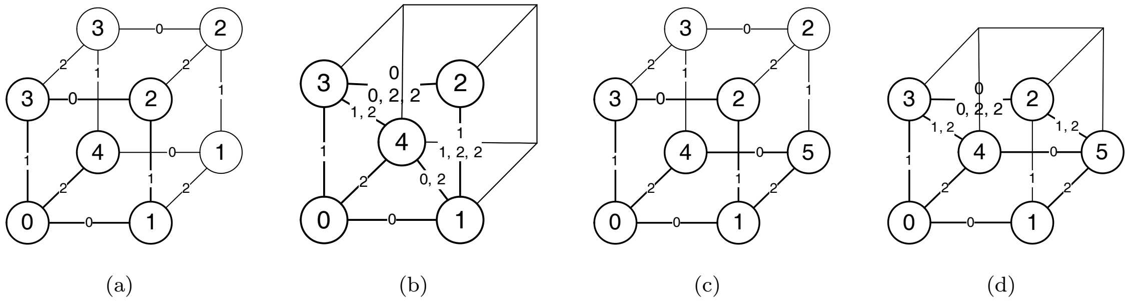
Start At the beginning, only peer 0 is active.

Step a Peer 0 is contacted by node 1 which wants to join the peer to peer network. Peer

0 integrates peer 1 as 0–neighbor since it does not currently have any other neighbor: The peers establish a link between each other which is tagged with the neighbor set $\{0\}$. Generally, a peer integrates a joining peer on its first vacant neighbor level, the neighbor levels are ordered such that lower neighbor levels always come first.

Step b Peer 2 contacts one of the two peers (here, it contacts peer 1) to join the network. The first vacant neighbor level of peer 1 is 1 since it already maintains a 0–neighbor, peer 0. Essentially, peer 1 opens up a new dimension for the hypercube, as depicted in figure 2.16(b). Peer 1 becomes the so called integration control node for the complete integration of peer 2 into the network: It is responsible for providing peer 2 with all necessary links - at the end of the integration process, peer 2 has to have neighbor links connecting it all currently existing neighbor levels, in order to be able to carry out complete broadcasts. Since peer 1 currently has two neighbors, a 0– and a 1–neighbor, it knows that it has to provide peer 2 with a 0– and a 1–neighbor, too. Peer 1 itself has become peer 2's 1–neighbor. Since there is currently no alternative, peer 1 selects peer 0 as the new 0–neighbor for peer 2. However, peer 0 can only become a temporary 0–neighbor for peer 2 because it already has another 0–neighbor, namely peer 1 - and it was said before that a peer can only have one neighbor per neighbor level. Essentially, peer 0 now covers a vacant position in the hypercube, i.e. it acts as if it occupies two positions in the hypercube, as depicted by the thin copy of peer 0 in figure 2.16(c). To mark the link between peers 2 and 0 as temporary relationship, it is tagged with the link set $\{0, 1\}$ (instead of $\{0\}$): This link set denotes the path from peer 2 via the position at which the link set is originally aiming to peer 0, the peer which currently covers this position. (This path is also well visible in figure 2.16(c)) Temporary link sets are always constructed by this rule.

Step c Peer 3 wants to join the network. Three cases are compared, viz. peer 3 contacting peer 0, 1 or 2 to join the network. In case peer 3 contacts peer 0 to join, peer 0 follows the general rule to integrate the peer on its first vacant neighbor level - which is 1, since peer 0 has a 0–neighbor, but no 1–neighbor. As its new 1–neighbor, peer 3 will now cover the temporary position that peer 0 used to maintain in the hypercube: Hence peer 0 can pass on links that are associated with this position to peer 3. Due to the construction rule of edge labels for temporary link sets, peer 0 is able to determine that link $\{0, 1\}$ to peer 2 is a link that is to be passed on to peer 3. Peer 3 then establishes a link tagged by link set $\{0\}$ to peer 3, as depicted in figure 2.16(d). In case peer 3 contacts peer 2 to join, peer 2 decides to integrate peer 3 as its new (and non-temporary) 0–neighbor. However, it does not carry out the integration itself: Since peer 0 currently covers the position that will soon be occupied by peer 3, the integration control responsibility has to be forwarded to peer 0. Peer 2 can do so via peer 0. Note that it is always possible for peers in the network to reach the node to which they have to forward the control integration, if necessary, in one hop (proved this in [106]). Peer 0 carries out the integration just as described above, arriving at figure 2.16(d). In case peer 3 contacts peer 1, peer 1 will integrate peer 3 on neighbor level 2, i.e., it opens up a new dimension for the hypercube. This leads to a momentary misbalance in the hypercube with some peers maintaining more links than others. However, the hypercube will only become misbalanced in the long run if there are 'joining hotspots' in the network. Burst joins of peers are no problem, the structure will balance itself again in the long run. Moreover, the information on vacant position in the structure is always spread in the network, i.e. it is likely that a joining peer will contact a network peer that has a vacant position to fill, inherently balancing the graph. In extreme cases, active balancing (for example, by sending joining nodes on a random walk through the network) has to be carried out. We defer this work to a future paper.

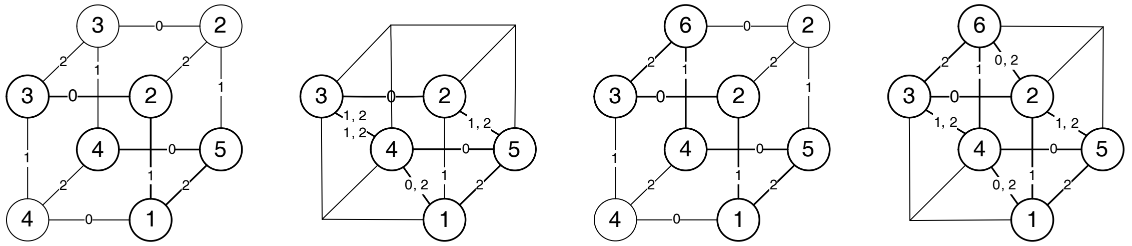


(a) (b) (c) (d)

Figure 2.17: Network topology construction continued

Step d Peer 4 arrives and contacts peer 0. Now, the network crosses a threshold - a hypercube with 2 dimensions cannot accommodate 5 peers, hence a third dimension is opened up. Peer 0 integrates peer 4 on its first vacant neighbor level as its new 2–neighbor. Peer 4 needs 3 neighbors, one on each neighbor level - but neither peer 0's 1–neighbor, peer 3, nor peer 0's 0–neighbor, peer 1, are linked to their own 2–neighbor which they could provide as a new neighbor to peer 4. Thus, peer 3 acts as temporary 1–neighbor for peer 4, whereas peer 1 acts as temporary 0–neighbor for peer 4, indicated once again by the link sets $\{0,2\}$ and $\{1,2\}$ among these peers (see figure 2.17(b)). Figure 2.17(a) shows the existing peers in the network in bold style and the positions that are additionally covered by them in thin style. Once again, note that the positions that are additionally covered by peers determine the temporary connections the peers have to maintain, plus their edge labels. Figure 2.17(a) also demonstrates another basic rule: Peers that are 'closest' to a vacant position in the hypercube structure are always chosen to cover it. Here, 'closest' means that the peer on the highest neighbor level to a vacant position covers it. (In the more complicated case when a peer has to cover several positions, a peer covers the power set of its vacant neighbor levels - however, refer to [106] to find a detailed discussion.) Among the other peers in the network, adding another dimension to the graph means the multiplication of existing links, too: For example, peers 1 and 2 could now both integrate 2-neighbors, which would then be linked on neighbor level 1. Thus, they tag their already existing $\{1\}$ link additionally as $\{1, 2, 2\}$ link. So do peers 2 and 3 with their already existing $\{0\}$ link.

Step e Peer 1 is contacted to integrate the newly arriving peer 5. Peer 1 is still lacking a 2–neighbor, thus peer 5 will be integrated on this position (figure 2.17(d)). Now, peer 1 can get rid of its $\{1, 2, 2\}$ link to peer 2: It is moved to peer 5. However, since 2 is not peer 5's final 1–neighbor either, the link stays temporary: Peers 2 and 5 now maintain a $\{1, 2\}$ link among them. Peer 5 takes over peer 1's temporary $\{0, 2\}$ link to peer 4, which still lacks its final 0–neighbor. It has found one now, namely peer 5.



(a) (b) (c) (d)

Figure 2.18: Network topology construction continued

Step f Let us assume that peer 0 suddenly leaves the network. In the maintenance protocol, it is obliged to carry out a peer removal process: Basically, it decides which existing peers that it knows will be chosen to take over responsibility for the positions it gives up. In our example, peer 0 leaves only one position vacant, its original position in the graph - however, a node which covers multiple positions will have to find successors for each of its positions in the graph. Peer 4 takes over peer 0's position, establishing temporary links to the former neighbors of peer 0, peers 1 and 3. Figure 2.18(a) shows the new distribution of covering responsibilities, figure 2.18(b) depicts the link structure that arises from this network state.

Step g Peer 4 is contacted by peer 6 and decides to integrate it as its new 1–neighbor. This position is currently covered by peer 3, hence peer 4 forwards the integration control to peer 3, just as described in step c. In the example, all temporary links that are currently owned by peer 3, but originally belong to the new position of peer

6, are restored and passed on to peer 6. Additionally, peer 3 integrates peer 6 as its new 2–neighbor, arriving at figure 2.18(d). Note that both in step f and g a joining peer could have contacted any peer in the network without misbalancing the graph structure since all peers maintain temporary links.

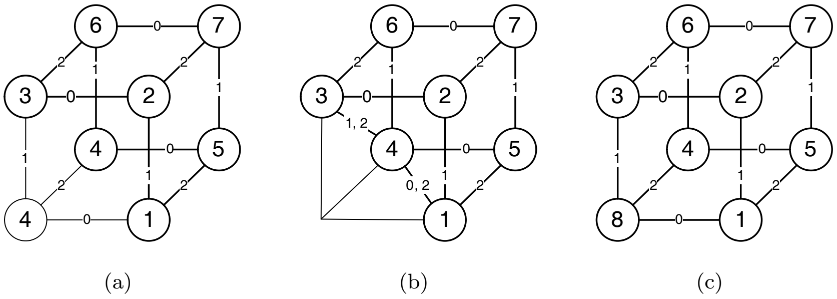


Figure 2.19: Network topology construction continued

Step h Peer 6 is contacted by peer 7, leading to peer 7's integration as peer 6's new 0–neighbor. Figure 2.19(a) and figure 2.19(b) depict the state of the network: Almost all positions of a complete hypercube graph with 3 dimensions are held by active peers, only peer 4 still covers two positions in the hypercube.

Step i On integrating peer 8, peer 4 pushes its links $\{1, 2\}$ and $\{0, 2\}$ to its new 2–neighbor, arriving at a complete hypercube topology again.

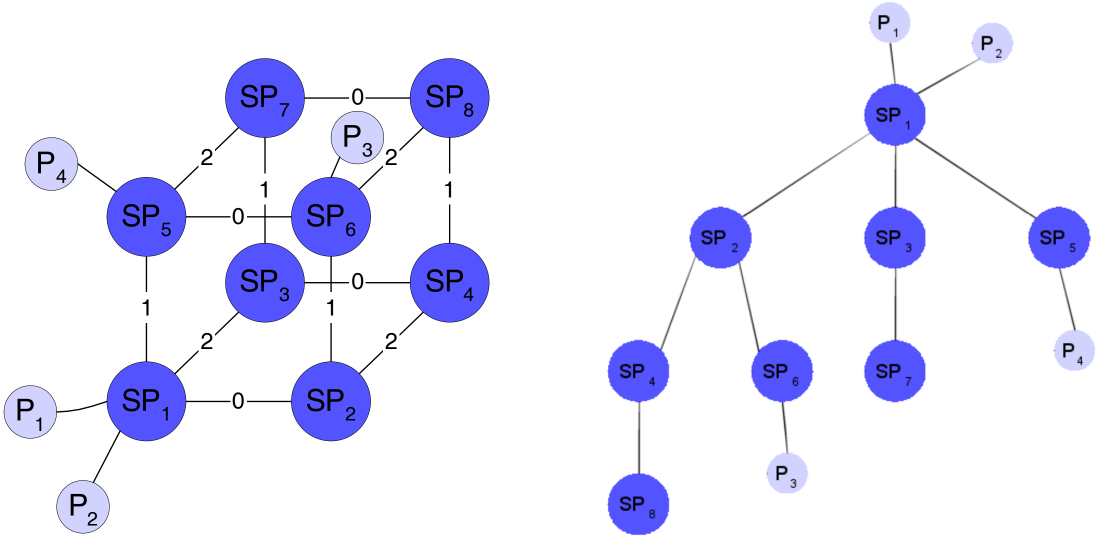
Link failures A link failure in the network leads to a node's immediate departure from the peer to peer topology, not being able to send any departing messages. If that happens, the topology must be able to recover and head back to a normal state. In the hypercube graph, one can always recover from a sudden node loss: The node that is closest to the vanished node (in terms of a metric called graph hop distance which uses the dimension order to compute a distance value between positions in the hypercube) contacts the vanishing node's neighbors by asking its own neighbors for them. The node then carries out the node departure routine on behalf of the vanished node.

2.2.2.3.3 Super Peers in a HyperCuP Topology A HyperCuP empowered peer to peer network features good scalability and search times as presented in the previous section.

The presented ideas of super peers and HyperCuP can be merged together, the resulting infrastructure is a so called super peer topology, where peers connect to super peers that build up the routing backbone for the whole network.

The responsibility of a super peer is the efficient query and answer routing as well as the distribution and execution of query plans based on local routing indices at each super peer.

It is an important decision is how to arrange the super peers in order to optimize the routing of queries in the network. It has been proven and practically shown in the Edutella peer to peer network [91] that super peers can be organized in the HyperCuP topology.



(a) Hypercube of dimension 3

(b) Spanningtree

Figure 2.20: A Hypercube and its spanningtree

Figure 2.20 shows a HyperCuP and its spanning tree (where some peers are added). The algorithm works as follows: All edges are tagged with their dimension in the hypercube. A super peer invoking a request sends the message to all its neighbors, tagging it with the edge label on which the message was sent. Neighboring super peers receiving the message forward it only via edges tagged with higher edge labels.

Peers themselves do not connect to each other in super peer networks. Instead they connect to a super peer thus forming a star like local network at each super peer. In turn the super peer takes over specific responsibilities for peer aggregation, query routing, etc.. This topology allows the peers to use their resources more efficiently while the super peer provides the necessary bandwidth and processing power to enable efficient and reliable query processing and answering.

Super peer based peer to peer infrastructures usually exploit a two phase routing architecture, which routes queries first in the super peer backbone, and then distributes them to the peers connected to the super peers. The last step can sometimes be avoided if the super peers cache data from their connected peers. Super peer routing is usually based on different kinds of indexing and routing tables, as discussed in [41].

CHAPTER III

Search in Peer to Peer Networks

In the previous chapter peer to peer technology was discussed. Techniques, methods and algorithms for storage and routing were presented. This chapter presents more sophisticated approaches that do not only take simple lookup of objects in peer to peer networks into account, but concentrates on search based on content.

The first sections introduce background information from the area of information retrieval and appropriate methods for both centralized and decentralized systems. Afterwards, approaches for information retrieval in peer to peer systems are discussed and challenging aspects conclude this chapter.

3.1 Background: Information Retrieval

The storage and retrieval of data or information are not tasks which were introduced by computer scientists. Since there are information, people want to make them persistent and save for the future for several reasons.

In a strict sense there is a distinction between the notions ‘data’, ‘information’, and ‘knowledge’ [12]. The distinct understanding of the terms depends on the area in which they are used (e.g. Shannon Weaver model of communication). Since it is not important to further distinguish the term in the context of this thesis, data and information will be used interchangeable; the notion ‘knowledge’ is understood as something that build upon data or information and thus is ignored in the following.

It is broadly accepted that the terms data and information are used interchangeable in the area of information retrieval, if it is not explicit necessary to distinguish. The terms ‘data retrieval’, ‘information retrieval’, and ‘document retrieval’ are also often used interchangeable, which is not precise enough in the context of information retrieval. The two last mentioned notions almost are used in the same sense and thus in this thesis they are also used in the common understanding:

Information retrieval (or document retrieval) is concerned with methods for storing, organizing and retrieving information from a collection of documents. In contrast, data retrieval is understood as searching based on attributes, e.g. in a database.

This means, information retrieval techniques deal with documents: ’Information retrieval is best understood if one remembers that the information being processed consists of documents.’ [104]

Baeza-Yates et al. define a document as ’a unit of retrieval. It might be a paragraph, a section, a chapter, a Web page, an article, or a whole book.’ [18]

Some definitions of information retrieval are:

- Information retrieval is the science and art of locating and obtaining documents based on information needs expressed to a system in a query language. Robert M. Losee [79]
- Part of computer science which studies the retrieval of information (not data) from a collection of written documents. The retrieved documents aim at satisfying a user information need usually expressed in natural language. Ricardo A. Baeza-Yates [18]
- Recall from storage of item of non-numerical information. Calvin N. Mooers [88]
- Information Retrieval deals with uncertainty and vagueness in information systems. IR Specialist Group of German Informatics Society, 1991

Satisfying information needs is a challenge for more than 4000 years; data is stored and must somehow retrieved if needed (see figure 3.1). People want to make their knowledge persistent and searchable. Far away from any elaborated approach, starting with papyrus, later on books were and still are the typical ways to store (textual) information.

New methods were created to be able to cope with the steadily increasing amount of information. One of the earliest and still important and prominent technique is the index: A set of words or concepts is associated with their origin (i.e. documents, book). Over time some ideas led to improved handling of the written information. Those ideas survived into the current technologies (Pages, chapter, indexes, table of content, etc.), but still is was all done manually[1].

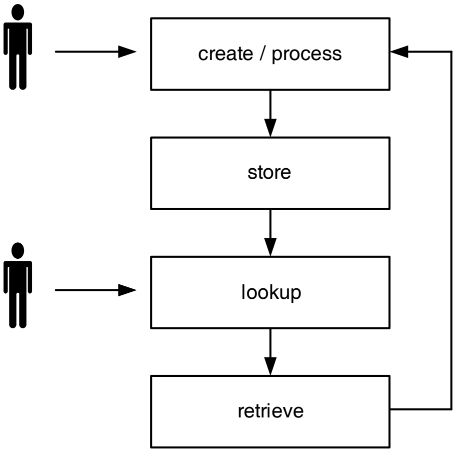


Figure 3.1: Information processing

After World War 2 the available computer systems and the increasing amount of information lead to the new research area, called information retrieval. Mainly two persons proposed ideas for that what was later called information retrieval by Calvin Mooes [88] in 1950: Vannevar Bush and Warren Weaver, who are sometimes called 'fathers of information retrieval'.

Regarding the vision in computer based information management, Vannevar Bush had many innovative ideas which he presented 1945 in his article 'As we may think' [33]. Ignoring the hardware aspect in his publication, he already presents ideas for computer systems used in business processes, depicts the notion of desktop computers and the presented Memex system already has ideas which were realized with hypertext and the world wide

[1] even today there are people creating premium indexes manually, i.e. http://www.indexers.org.uk

web.

Furthermore, he discusses ideas of the modern information retrieval. On the one hand he presents ideas for manage and retrieve documents like books, newspapers, etc. – but he also mentions ideas which can be seen similar to logic reasoning and associative storage: ‘Thus science may implement the ways in which man produces, stores, and consults the record of the race. It might be striking to outline the instrumentalities of the future more spectacularly, rather than to stick closely to the methods and elements now known and undergoing rapid development, as has been done here.’.

It is interesting that the term information retrieval (IR) was introduced in 1950 by Calvin Mooers [88] into the literature of documentation (subject access to information, and how to index scientific articles and reports).

To that time, the idea that an information retrieval system should be judged by two measures – how well it captures relevant documents, and how well it rejects the irrelevant – was only given implicit and not clearly stated. The formalization was done by Kent et al. in 1955 [69] who defined the two measures ‘recall’ (proportion of relevant documents that are retrieved) and ‘pertinency factor’ (proportion of retrieved documents that are relevant). The latter measure was later called ’relevance ratio’ and the, about mid of 1960, ’precision ration’ (or precision) became standard.

3.1.1 History

The area of information retrieval is typically divided into two periods [65]: First period 1955 – 1975, and second period starting 1970. The first period is characterized by fundamental research and research in the area of key technologies. The second period focuses on the enhancements of existing approaches and their application in diverse areas (this is the reason why there is an overlap of five years). A detailed discussion can be found in the work of Michael Lesk [73].

The growing number of machine readable documents and the network based communication stimulated research in the area of information retrieval (the automation starting in the 1950s with punchcards).

The first benefit that information retrieval entailed was the possibility to use boolean operation for searches on previously indexed terms; This is called postcoordination. Unlike precoordination where terms are associated with topics manually in advance (e.g. by a librarian), postcoordination allows users to indicate the topic using a combination of descriptors (indexed terms).

The main processes in information retrieval are indexing and searching. In the beginning, indexes were created manually and the focus was on the search process. This means all methods which can be used to check whether a term is the index or not and how the query is compared to the index terms. The next step was the automated indexing process, which had two goals: 1) To minimize the effort for the indexing process, and 2) to improve the representation of the content of the documents. Luhn[2] is one of the pioneers and discusses in his work ([85]) the possibilities of automated deriving of index terms from machine readable documents.

Boolean logic was and still is very important for databases and information retrieval systems[3]; term are evaluated using boolean operators. The results set then contains all documents which satisfy the boolean operation. Even though extended methods like proximity search (search term should be located near each other in the document) or truncation search (ignore suffixes) emerged (at least in theoretical work), the boolean model kept its prominent role. Nevertheless, the main drawbacks using boolean search are

- For users it is not easy to formulate a query since the terms must be chosen in such a way, that the information need is represented; this requires some routine.
- Without exactly knowing what's in the document collection it is a priori not possible for the user to estimate the quality and quantity of the results.
- The document collection is simply divided into two groups: One are the documents which satisfy the query, and the other are the documents which do not satisfy the query. This implies that all hits have the same relevance for the user information need. Obviously, this does not reflect the reality, where some documents are more relevant than other regarding a particular query.

The drawbacks led to alternatives, like the vector space model or the probabilistic model, which will be introduced later in this chapter.

Other techniques which are still very important in modern information retrieval were introduced, e.g. measurements for the quality of results (i.e. precision and recall), the idea of relevance feedback (users give feedback into the system regarding a presented result set), basic clustering approaches, association mechanisms.

First implementations of information retrieval system were SMART [103], MEDLARS (Medical Literature Analysis and Retrieval System), and OCLC (Online Computer Library Catalog) [65].

Starting with the 1970s (and overlapping with the first period) the implementation of

[2] Following the calling 'Read before You cite!'[109] the author of this thesis cites Mr. Luhn with 'Hans Peter Luhn', instead of the typical citation 'N. P. Luhn', which was maybe caused by an OCR-error turning the 'H' into a 'N'.

[3] the boolean algebra as a mathematical algebraic structure was published by George Boole in 1847, 'The mathematical analysis of logic'.

information retrieval systems was further promoted, which was driven by the increasing number of machine readable documents. As milestones of the second period can be considered Fuzzy Set, Probabilistic Ranking Models, Improved Relevance Feedback, Bayesian Networks, Inference Network Models, and the Belief Network Model.

3.1.2 Information Retrieval

In early years, computers were used to do mathematical calculations. Text processors, graphics and more sophisticated applications came later. Thus, information retrieval was understood as retrieval of numerical data.

Information retrieval is one of the two main communities that are working in the area of finding information. The database community has a string background as well, but differs inherently from information retrieval; this will be discussed in the next section, before getting to the modeling of information retrieval systems.

3.1.3 Databases and Information Retrieval

Databases and information retrieval have the same roots[4]. In both areas there is the challenge of finding information stored in a system. In this section the similarities and differences are discussed; this leads to the understanding of concepts in information retrieval, e. g. the notion of 'relevance'.

The key concept of databases is that all data are represented a mathematical relations [39]. The model provides a simple, yet rigorously defined, concept of how users perceive data. In the relational model, a database is a collection of two-dimensional tables. Such a relational table is composed of a set of named columns and an arbitrary number of unnamed rows. The columns of the tables contain information about the table. The rows of the table represent occurrences of the 'thing' represented by the table. A data value is stored in the intersection of a row and column. Each named column has a domain, which is the set of values that may appear in that column. The organization of data into relational tables is known as the logical view of the database. That is, the form in which a relational database presents data to the user and the programmer. The way the database software physically stores the data on a computer disk system is called the internal view (which depends on the concrete implementation of a product).

In the following a short example is given. First, some tables are defined which contain information about books, authors and publishers (in the following only some basic SQL

[4] Restricted to *relational* databases here, since they are the most comparable ones; other types like hierarchial databases are not important in this context. Furthermore, the relational model can be extended for other purposes without losing its fundamental principles, i.e. with object oriented features [47]

statements for data retrieval are used).

The first table has attributes for books: ISBN, title, author and publisher (for simplicity, data types are used).

book	ISBN	title	author_ID	publisher_ID
	088175188X	Principles of database [...]	01	01
	0127082409	Theoretical Studies in Computer Science	01	02
	0914894951	Computational aspects of VLSI [...]	01	01
	0387962549	Disquisitiones Arithmeticae	02	03
	0895792524	3:16 Bible Texts Illuminated	03	04
	0201038129	Surreal Numbers	03	05
	080650711X	World As I See It	04	06

Table 3.1: Book table

The first two attributes contain concrete values, whereas author and publisher link to other tables. Those references are used to 'normalize' tables, which means that as few as possible attributes should be placed in one table to avoid redundancy.

publishing_Company	publisher_ID	name
	01	Computer Science Press
	02	Academic Press
	03	Springer
	04	A-R Editions
	05	Addison-Wesley Professional
	06	Citadel Press

Table 3.2: Publisher table

author	author_ID	aname
	01	Jeffrey D. Ullman
	02	Carl F. Gauss
	03	Donald E. Knuth
	04	Albert Einstein

Table 3.3: Author table

The three tables can now be used to answer queries using the Structured Query Language (SQL), which used to create, modify and retrieve data from relational database management systems.

A simple SQL query has three parts. The 'SELECT' part defines which columns will be put in the results set. 'FROM' is used to indicate which tables the data is to be taken from, and 'WHERE' is used to define the conditions for each row. The result set then contains the column values of all rows that satisfy the constraints.

Which title has the book with ISBN 0387962549?

```
SELECT title FROM book WHERE ISBN = 0387962549
```

results	title
	Disquisitiones Arithmeticae

Figure 3.2: A simple SQL query and the resulting table

A join combines records from two or more tables to place more complex requests to the database.

Which books wrote ‘Jeffrey D. Ullman’?

```
SELECT title
FROM books NATURAL JOIN author
WHERE books.author_ID = author.author_ID
      AND aname = ‘Jeffrey D. Ullman’
```

results	title
	Principles of database [...]
	Theoretical Studies in Computer Science
	Computational aspects of VLSI [...]

As stated above in the area of information retrieval the user should get the best answer regarding a query. This means only a limited number of relevant hits should be put in the result set which is presented to the user.

This implies that results are not 100% correct, but somehow fit in the result set, which is contraire to the understanding of the database community. An SQL-query gets results which match exactly, or the result set is empty. Typos or tiny differences in a literal result in unanticipated result sets.

Although it is concerned with finding complex data, data warehousing is not information retrieval. The queries which are used in data warehousing are posed against a number of databases, and thus work on fields, which defines data retrieval.

3.1.4 Modeling

As depicted in figures 3.3 and 3.4 an information retrieval system processes data and queries as inputs and uses several methods to process the inputs.

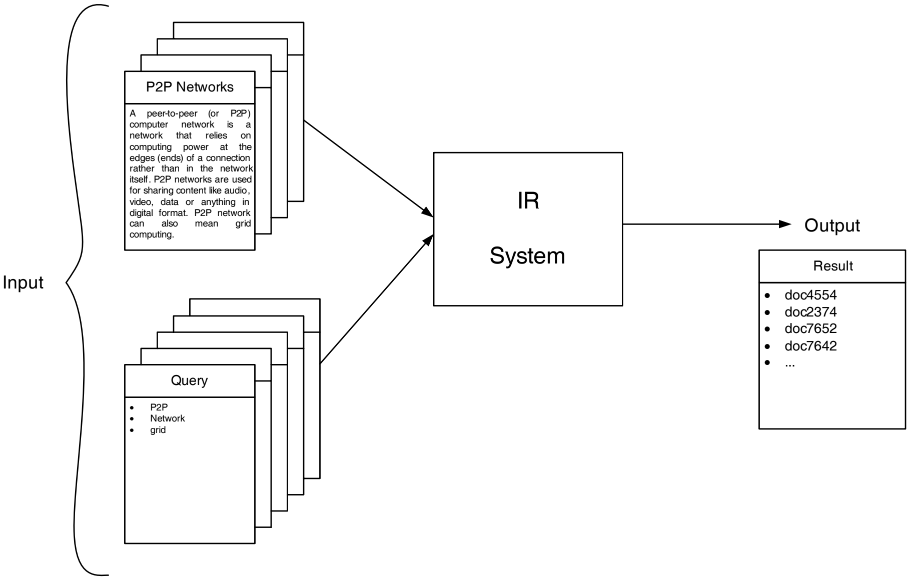

Input
P2P Networks
A peer-to-peer (or P2P) computer network is a network that relies on computing power at the edges (ends) of a connection rather than in the network itself. P2P networks are used for sharing content like audio, video, data or anything in digital format. P2P network can also mean grid computing.
Query
P2P
Network
grid
IR
System
Output
Result
doc4554
doc2374
doc7652
doc7642
...

Figure 3.3: Information retrieval system

The following section describe the parts of a typical information retrieval systems and thus introduces the model of such a system.

With respect to figure 3.4 the parts from left to right are discussed in the following. The complete process can vary from system to system. Not all steps will be found in each system; as figure 3.4 shows, processing parts can be skipped.

Information retrieval works mainly in the field of efficiently creating an index and process queries against the index. Thus, in the following sections the techniques and models used are introduced. Methods which do not rely on an index but do sequential searching (e.g. algorithms from Knuth, Morris and Pratt or the family of Boyer-Moore algorithms) are not further considered here. They are mainly used in situations, where indexes cannot be built and sequential searching are the only way to obtain data from a document collection, e.g. online processing of documents.

3.1.4.1 Building the Index

3.1.4.1.1 Structure Recognition This is a very substantial step which splits the documents into components that belong together in the sense of structural relations, e.g. paragraphs and sentences, but also structural elements like links between documents or within one document (inter- and intra-links.)

The area of structural text analysis is a parallel branch in the area of information retrieval research and goes far beyond the scope of this thesis.

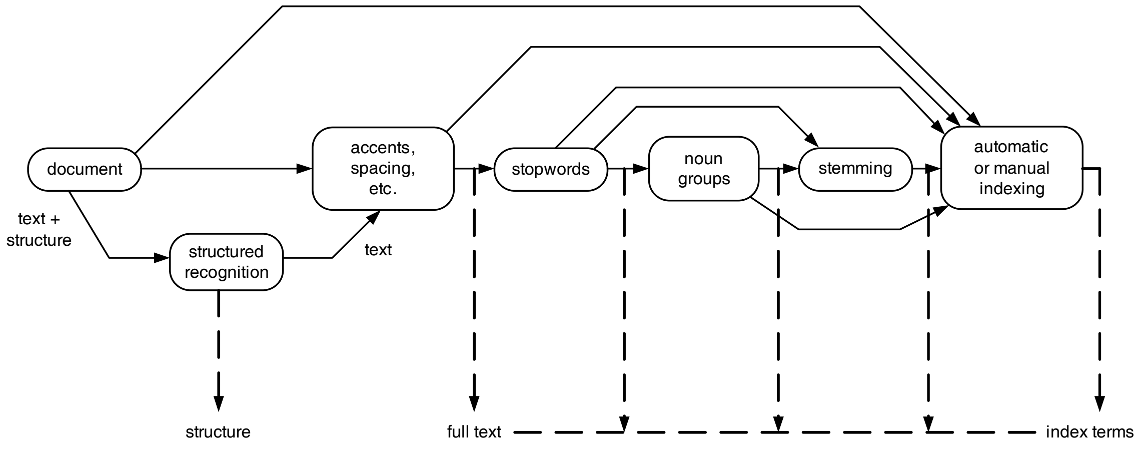


Figure 3.4: Information retrieval process

3.1.4.1.2 Word Recognition Text documents consist of characters and symbols. The characters form words and the words are separated by the symbols. Typically the words are built using the alphabet 'a' – 'z', 'A' – 'Z', '0' – '9', and country-specific characters, e.g. the German umlauts 'ä', 'ß', etc. The symbols are used as whitespaces and delimiters (punctation).

The task of word recognition is to remove all symbols which are not part of words. This steps reduces the amount of text which has to be processed in the next steps. This process is also called splitting or tokenizing the text document. The resulting set of words is often called bag of words in information retrieval.

3.1.4.1.3 Stop Words Every language (spoken or written) has a number of words which occur very frequently. Those commonly used words are deemed irrelevant for searching purposes, because they appear in nearly every sentence and thus cannot used to discriminate.

The list defining the words which are removed from the text is call stop word list. Those lists are given for almost every language and is a further step to save both space and time, since the set of words gets smaller and thus less words need to be compared later when querying. Typical words in a list used with English texts would be 'and', 'or', 'in' and 'the'.

3.1.4.1.4 Noun Groups The notion 'Noun groups' means two different things which are used independently and the exact meaning depends on the context.

First, noun groups are used to further minimize the number of words by eliminating verbs, adjective etc. This is based on the observation that user mainly query for nouns. In this step, sometimes thesauri are used to aggregate nouns to noun groups. The group of nouns which can be used best to classify a document are called proper nouns.

The second area which is titled 'noun groups' works with natural language processing, i.e. evaluating queries which are posed as questions. A query is then syntactically and semantically analyzed with the goal to set the information keys (nouns) in the query in relation to each other.

3.1.4.1.5 Stemming The two previous steps 'Stop words' and 'Noun groups' already shortened the list of words to process.

In most cases, morphological variants of words have similar semantic interpretations and can be considered as equivalent. For this reason, a number of so called stemming algorithms (or stemmers) have been developed.

Stemming means to apply rules to words, which reduce to the principal form. The implementation of a stemmer always needs to take into account for which natural language it is written. English stemmers are fairly trivial, but nevertheless there are some problems with ambiguity, e.g. 'dries' is the third-person singular present form of the verb 'dry' and not only an suffix added to the root of a word.

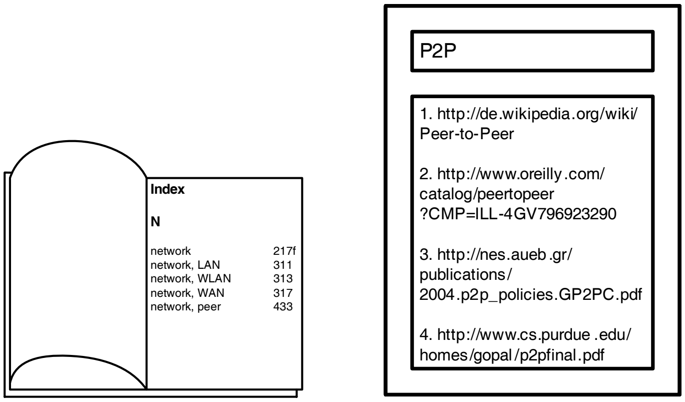
The most commonly used stemmer algorithm was developed by Martin F. Porter in 1980 [5], who gave the name: Porter stemming algorithm, or Porter Stemmer for short. Other stemming algorithms are Paice/Husk Stemming [96], Lovins Stemming Algorithm [82], Dawson Stemming Algorithm [49] and Krovetz Stemming Algorithm [71].

Today, a programmer can resort to many existing implementation of stemmer, but nevertheless there are some challenges left, which less come from the natural languages, but from technical aspects in the implementation, i.e. coding in unicode, UTF8, etc.

3.1.4.1.6 Index In previous sections the term 'index' is already used informally as a list of words, where each word is assigned a list of occurrences of the word in the documents.

An index basically is a data structure over the text. It allows to minimize the space required and to speed up the searches. An inverted index is composed of two elements: the vocabulary and the occurrences. The former is a set of lists. For each word of the vocabulary one occurrence list has all positions where the word occurs. These positions can refer to words in one document (well know from the index in books, see figure 3.5(a)) or to words in several document (see figure 3.5(b)).

The idea of an index is very straightforward and easy to understand. Even if methods like stemming, stop words etc. are applied, there is still room for improvement, i.e. index

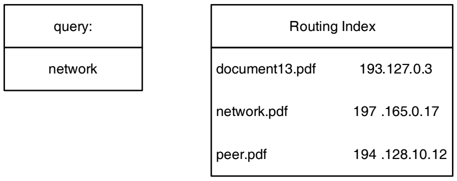

Index
N
network 217f
network, LAN 311
network, WLAN 313
network, WAN 317
network, peer 433
P2P
1. http://de.wikipedia.org/wiki/Peer-to-Peer
2. http://www.oreilly.com/catalog/peertopeer?CMP=ILL-4GV796923290
3. http://nes.aueb.gr/publications/2004.p2p_policies.GP2PC.pdf
4. http://www.cs.purdue.edu/homes/gopal/p2pfinal.pdf

(a) Word index (book) (b) Inverted index (documents)

Figure 3.5: Indexes

compression or alternative storage methods like suffix trees or signature files [133].

Another use of indexes is not coming from the information retrieval, but very important in this thesis: An index can also contain routing information. Given a network, some nodes store so called routing indexes which contain for some information which nodes are most prominent to receive e.g. a query. An example is shown in the figure on the right. In the index there is an entry for the query containing the term 'network'. For this keyword there is a list of pairs (document, address), which can be used to forward the query to the promising peers (=addresses) only, because in a previous step the document listed was in the result set for the particular query. This basic idea is used and discussed in detail in the algorithm which is presented in chapter IV.



query:
network

Routing Index	
document13.pdf	193.127.0.3
network.pdf	197 .165.0.17
peer.pdf	194 .128.10.12

Figure 3.6: Routing index

3.1.4.2 Working with the Index

At this point, the way of information into the information retrieval system is introduced. Each part which processes data is discussed; this defines the processing of inputs (documents and queries).

3.1.4.2.1 Query Models There are several query models, which evolved over time. The following sections present both the so called classic models and shortly models that are related to modern information retrieval.

Boolean Retrieval Boolean retrieval is historically the first retrieval model. In the early day, boolean retrieval was the only method that was feasible, because memory was not a rare resource and for documents stored using magnetic tapes the decision whether it should considered as a result or not was to be taken as soon as possible.

The boolean model is a simple retrieval model based on set theory and boolean algebra. Queries specified as boolean expressions have precise semantics. Because of its inherent simplicity and neat formalism, the boolean model received great attraction and was adopted by many information retrieval systems, e.g. library OPACs.

Documents and queries are represented as sets of index terms (set theoretic approach). For the boolean model, the term index weights are all binary, i.e. $w_{i,j} \in \{0,1\}$. A query Q is a conventional boolean expression. Let q_{dnf} be the disjunctive normal form for the query q[5]. Further, let q_{cc} be any of the conjunctive components of q_{dnf}. The similarity of a document d_j to the query q is defined as:

$$sim(d_j, q) = \begin{cases} 1 & \exists q_{cc} \in q_{dnf} \quad | \quad g_i(d_j) = g_i(q_{cc}) \quad \forall k_i \in q_{cc} \\ 0 & \text{otherwise} \end{cases}$$

Obviously, no ranking of results can be done, since there is no gradually score, but only a binary decision whether a document satisfies a query or not. The documents in the result set are only exact matches without a ranked order or limitation in quantity.

Ranked Retrieval In one of Luhn's [85] early papers he states: 'It is here proposed that the frequency of word occurrence in an article furnishes a useful measurement of word significance. It is further proposed that the relative position within a sentence of words having given values of significance furnish a useful measurement for determining the significance of sentences. The significance factor of a sentence will therefore be based on a combination of these two measurements.'

This quote fairly summaries Luhn's contribution to automatic text analysis: His assumption is that frequency data can be used to extract words and sentences to represent a document.

Zipf's Law: Let f be the frequency of occurrence of various word types in a given position of text and r their rank order, that is, the order of their frequency of occurrence, then a plot relating f and r yields a curve similar to the hyperbolic curve in figure 3.7. This is in

[5] A query q is composed of index terms linked by three connectives: not, and, or. Thus, a query can be represented as a disjunction of conjunctive vectors, i.e. in disjunctive normal form (DNF). For instance, the query $q = t_a \wedge (t_b \vee \neg t_c)$ can be written in DNF as $\vec{q_{dnf}} = (1,1,1) \vee (1,1,0) \vee (1,0,0)$, where each of the components is a binary weighted vector associated with the tuple (t_a, t_b, t_c) and are called the conjunctive components of $\vec{q_{dnf}}$.

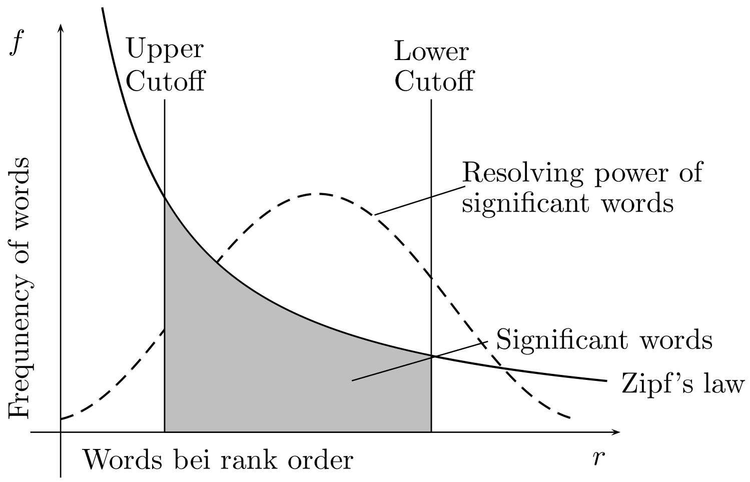


Figure 3.7: Word frequency diagram (adapted from [84])

fact a curve demonstrating Zipf's Law [7], which states that the product of the frequency of use of wards and the rank order is approximately constant.

Zipf distributions are ubiquitous in content networks, the Internet and other collections. The distribution was first discovered by the Harvard linguistic professor G. K. Zipf, and has become one of the most empirically validated laws in the domain of linguistic quantities. If we count the number of times each word appears in a text (called frequency) and assign each word a rank based on its frequency (i. e. rank=1 is the word that appears the most), we can see that the product frequency rank for each word is roughly equal to a constant. In a more general context, this corresponds to the observation that the frequency of occurrence of an event as a function of the rank is a power-law function. Recent research has discovered this distribution as a typical distribution of information on the Internet [13, 14, 51, 38, 86]. In peer to peer networks typical consumers are interested in only subsets of all available content and content categories [42], or as Scott Shenker says in his paper: 'People are looking for hay, not for needles!' [37]. Documents are also distributed following Zipfs law, i.e. many consumers are interested in resources which are held by a few providers.

Luhn used it as a null hypothesis to enable him to specify two cut-offs, an upper and a lower (see figure 3.7), thus excluding non-significant words. The words exceeding the upper cut-off were considered to be common and those below the lower cut-off rare, and therefore not contributing significantly to the content of the article. He thus devised a counting technique for finding significant words. Consistent with this he assumed that the resolving power of significant words, by which he meant the ability of words to discriminate content, reached a peak at a rank order position half way between the two cut-offs and from the peak fell off in either direction reducing to almost zero at the cut-off points. A certain arbitrariness is involved in determining the cut-offs.

It is interesting that these ideas are really basic to much of the later work in information retrieval; this will become clear in the next sections, which introduce several metrics used in information retrieval.

Vector Space Model The vector space model removes the limitation of boolean weights. Documents and queries are represented as vectors in a t-dimensional space (algebraic approach).

The idea is to assign non-binary weights to index terms in queries and documents, and then based on that compute the similarity between a query and documents[6].

For the vector space model, the weight $w_{i,j}$ associated with a pair (k_j, d_j) is positive and non-binary. Further, the index terms in the query are also weighted. Let $w_{i,q}$ be the weight associated with the pair (k_j, q) where $w_{i,q} \geq 0$. Then, the query vector is defined as

$q = (w_{i,1}, w_{i,2}, \ldots, w_{i,t})$, where t is the total number of terms.

The vector for a document d_j is represented by

$d_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$

The similarity of a query q and a document d_j is defined by

$$sim(d_j, q) = \frac{d_j \times q}{\|d_j\| \times \|q\|}$$

which is

$$\frac{\sum_{i=1}^{t} w_{i,j} \cdot w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \cdot \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

This can be thought as documents and queries in a t-dimensional vector space. The similarity is defined by the cosine of the angle between a documents and a query, value $\in [0, 1]$. Documents can be ranked according to their similarity; documents beyond a certain threshold can be delivered as results.

How to assign the weights? The basic assumption in the vector space model is the assignment of weights to the terms, allowing similarity measures with gradually relevance. A simple way to create weights is to count how often a term occurs in a document. This is based on the idea that a term is important if it occurs often in a document. This is called the *within document frequency*, and is denoted $f_{d,t}$ for term t in document d, or simply the

[6] As described above (section 3.1.4.1) information retrieval systems adopt index terms to index and retrieve documents, where an index term is simply any word which appears in the text of a document

term frequency, TF. A similarity measure then can simply be the inner product of a query and a document.

But this does not take into account term scarcity. If a word appears only in one document, it is a more important term than a word which appears in more documents (discrimination, f_t). This is means a term is weighted according to its *inverse document frequency*, IDF:

$$w_t = \frac{1}{f_t}$$

Combining the above, the idea can be stated as: Assign a weight which

1. increases with the number of occurrences within a document
2. increases with the rarity of a term across the document collection

which is the well known TFxIDF, which is the most used weighting in information retrieval:

Let N be the total number of documents in the collection and n_i the number of documents in which the index term k_i appears. Let $freq_{i,j}$ be the raw frequency of term k_i in the document d_j (i.e. the number of times the term k_i occurs in the text of document d_j). To avoid misbalancing resulting from long documents, the term frequency is normalized as

$$f_{i,j} = \frac{freq_{i,j}}{\max_w freq_{w,j}}$$

where $\max_w$ is computed over all terms w in a document d_j. If a term k_i does not occur in a document d_j then $f_{i,j} = 0$. Further, let idf_i be the inverse document frequency for k_i:

$$idf_i = \log \frac{N}{n_i}$$

The best known term weighting scheme TFxIDF uses weights which are given by

$$w_{i,j} = f_{i,j} \cdot \log \frac{N}{n_i}$$

The log is included to prevent a term for which $f_t = 1$ from being regarded as twice as important as a term for which $f_t = 2$.There is a family of TFxIDF schemes, which slightly differ in the factorization. The mentioned formula is the one which is mostly used.

Example: Assume three documents containing some word as follows:

1. D_1: peer, computer, network, filesharing, network
2. D_2: computer, machine
3. D_3: node, peer, network, peer

Ignoring the normalization in this simple example, the term frequencies, inverted documents frequencies and the resulting TFxIDF values are shown in the tables 3.4.

(a) Term frequencies

	TF_i		
terms	1.	2.	3.
peer	1	0	2
computer	1	1	0
network	2	0	1
filesharing	1	0	0
machine	0	1	0
node	0	0	1

(b) Inv. doc. frequencies

terms	IDF_i
peer	1.5
computer	1.5
network	1.5
filesharing	3
machine	3
node	3

(c) TFxIDF

	TF_ixIDF_i		
terms	1.	2.	3.
peer	1.5	0	3
computer	1.5	1.5	0
network	3	0	1.5
filesharing	3	0	0
machine	0	3	0
node	0	0	3

Table 3.4: VSM weight computation

Having a query with the query terms network and computer, figure 3.8 shows the vectors in a 2d-coordinate system. The angles between the queries and each document measures how similar the vectors (and thus the query and a document) are. Since $b < a$, document 1 matches the query better than document 3.

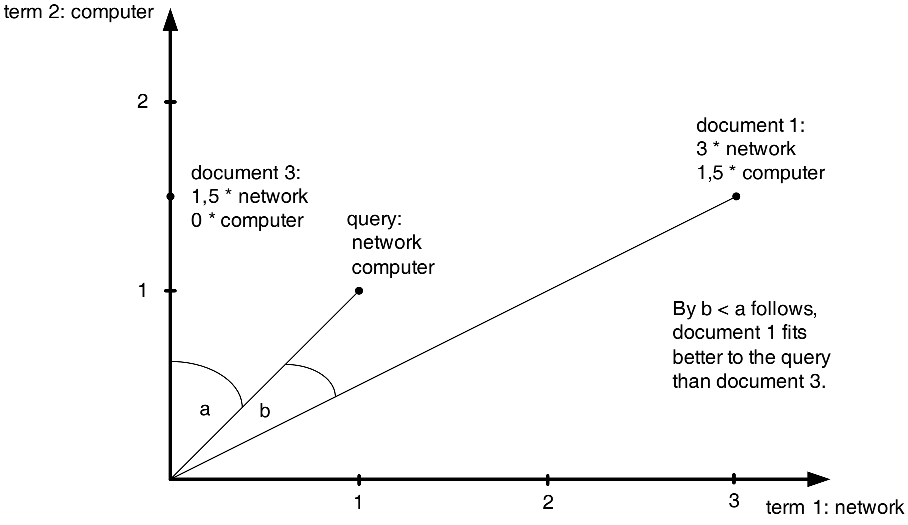


Figure 3.8: Vector space model

3.1.4.2.2 Other Models The information retrieval techniques described above use only a small amount of the information associated with a document as the basis for relevance decisions. Despite this inherent limitation, they often achieve acceptable precision because the full text of a document contains a significant amount of redundancy.

In information retrieval, there are several methods that try to capture more information about each document to achieve better performance.

Probabilistic model: Documents and queries are represented using probabilities. It is a formal model that attempts to predict the probability that a given document will be relevant to a given query. It ranks retrieved documents according to this probability of relevance (Probability Ranking Principle) and relies on accurate estimates of probabilities (Bayesian networks).

Natural Language Processing does complex syntactical and semantical analyses of the text to capture more information for result ranking. Queries are mostly complex question like 'How many people are associated with the Rotary Club in Germany?' instead of keywords.

Latent Semantic Indexing (LSI): In addition to recording which keywords a document contains, this method examines the document collection as a whole, to see which other documents contain some of those same words. LSI considers documents that have many words in common to be semantically close, and ones with few words in common to be semantically distant. This simple method correlates surprisingly well with how a human being, looking at content, might classify a document collection. Although the LSI algorithm doesn't understand anything about what the words mean, the patterns it notices can make it seem astonishingly intelligent.

3.1.4.3 Measurements / Evaluation

Talking about evaluation of an information retrieval systems requires consideration of the objectives of the retrieval system. Basically, any software system has to provide the functionality it was developed for; this is called a functional analysis, where each specified system functionality is tested one by one.

After showing that a system provides the conceived functionalities, the most common measures of system performance are time and space (both memory (RAM) and storage (e.g. gigabyte on a harddisk) used to yield the functionalities. The shorter the response time and the smaller the space used, the better is the system is considered to be (of course, there is an inherent tradeoff between space complexity and time complexity, which frequently allows trading one for the other). For an information retrieval system this means that e.g. index structures and communication channels are evaluated and improved if necessary and possible. Those measurements are summarized as *performance evaluation*.

On the other hand, for information retrieval systems there are other measurements, which are titled *retrieval performance evaluation*. Such an evaluation concentrates on how good results sets are. This means, how relevant the results are for the query posed by a user. Thus, information retrieval systems require the evaluation of how precise the answer set is [18].

Recall and Precision. Recall is the fraction of the relevant documents which has been retrieved – Recall is the fraction of (all) relevant material that is returned by the search.

Precision is the fraction of the retrieved documents which is relevant – a measure of the number of relevant documents in the set of all documents returned by a search. It forms a natural pair with recall.

In other words: $Precision = \frac{true\ positives}{(true+false\ positives)}$ and $Recall = \frac{true\ positives}{(true\ positives+false\ negatives)}$.

Interesting is the fact, that it seems to be accepted that relevance is additive. There is no research discussing the possibility, for example, that two irrelevant documents might become relevant if put together.

The problem of objective measuring. Since ‘relevance’ is a key concept in information retrieval, the challenge is to find a way to benchmark the results of an information retrieval system. The theoretic measures are recall and precision; obviously, they do not give an absolute number which can be directly compared to other results, but need some reference value to be compared to. Practically, such evaluations are usually based on test reference collections and on evaluation measures. The test reference collection consists of a collection of documents, a set of example information requests, and a set of relevant documents (provided by specialists) for each example information request. [18]

3.1.5 Summary and Conclusion

The next sections will introduce the area of retrieval in peer to peer systems. From the previous section it is important to understand which methods from information retrieval are state of the art and which components are needed. Especially, TFxIDF together with its inverted document frequencies are inthe focus of information retrieval in peer to peer networks, because it allows content based search using an efficient and well established approach.

3.2 Information Retrieval in Peer to Peer Networks

In the previous part of this chapter the background needed to discuss information retrieval in peer to peer networks was presented. In the following the challenges for information retrieval in peer to peer networks are introduced, followed by a discussion of state of the art systems and approaches for information retrieval in peer to peer. This chapter concludes with the open problems and thus motivates the development of the ProToRaDo algorithm presented in chapter IV.

3.2.1 Challenges

Besides the general challenges which arise from the peer to peer idea, ranked top-k ranking in peer to peer networks has to address four challenges:

Mismatch in scoring techniques and input data used by the different peers can have a strong impact on getting the correct overall top-scored objects. Since network should be minimized, but nevertheless integrate the top-scored objects from all different peers, each peer has to decide how to score answers to a given query.

Using only distributed knowledge and thus different input data to score answers complicates top-k retrieval, because many scoring measures that take global characteristics into account simply cannot be evaluated correctly with limited local knowledge. Joining and leaving peers influences the calculation further.

Minimizing network data transfer means that one should strive to only exchange the minimal amount of information necessary between peers. This is also relevant for merging result sets in peers, where one wants to minimize incoming data yet still produce a complete top-k answer list.

No continuous index updates. In peer to peer networks peers join and leave the network at unpredictable intervals. Top-k retrieval and routing has to take this into account without requiring continuous index updates limiting scalability of the algorithm. Obviously, volatility of the peers cannot be arbitrarily high, as no kind of statistics would be meaningful then.

3.2.2 Existing Systems / Approaches

During the last four years there have been reported several ongoing efforts attempting to produce satisfactory solutions for the field of highly distributed peer to peer information retrieval. Arguably the first project to demonstrate the potential of information retrieval in peer to peer system was the Infrasearch project [95]. Infrasearch was a Gnutella[18]-based meta-search engine which demonstrated the potential of peer to peer networking networking in highly diverse information environments. Subsequently, and after the Sun Inc. initiated JXTA[7] [60] project began, Infrasearch was acquired by Sun and was transformed into the JXTASearch [126] project.

In the following years the aspects of information retrieval in peer to peer networks were discussed in several approaches. The following sections give an overview of the state of the art of peer to peer information retrieval.

Aberer and Wu provide a good theoretical background in [11] where they present a ranking algebra as a formal framework for ranking computation. They show that not only one global ranking should be taken into account, but several rankings must be seen in different contexts. Their ranking algebra allows aggregating the local rankings into global rankings.

In the context of databases Agrawal et al. [15] propose several approaches to rank database query results, under the assumption that there is only one table, and only conjunctive queries are used. Instead of returning only complete matches, a similarity value (similarity between query and table row) is calculated. If a condition is matched by the row data, the frequency factor (analogous to inverse document frequency) is added to the similarity value. They present similarity-measures for numerical data and range conditions.

Bruno et al. [32] use a formalisation for the characteristics of retrieval in web databases, when several web accessible databases are used to create a top-k list, where each atom maybe answered from a different database.

3.2.2.1 PlanetP

PlanetP [43] was stated to be the first approach that supports content ranking in unstructured peer to peer networks.

It concentrates on peer to peer communities in unstructured peer to peer networks with sizes up to ten thousand peers. The authors clearly discuss in their analysis, that the used gossiping limits the algorithm to that number of peers.

In PlanetP two data structures for searching and ranking are introduced, which create a replicated global index, using gossiping. Each peer maintains an inverted index of its document and spreads the term to peer index. Based on this replicated index a TFxIDF-ranking algorithm is implemented.

As discussed in the previous sections, information retrieval systems use a global index containing term and inverse document frequencies. In a distributed environment this is

[7] http://www.jxta.org

not feasible. PlanetP solves this problem by maintaining a replicated network wide index which is updated using gossiping within the network.

The PlanetP system does not use collection wide information like e.g. the inverted document frequency of query terms directly, but circumnavigates the problem by using a so called inverted peer frequency (IPF). The inverted peer frequency estimates for all query terms, which peers are interesting contributors to a certain query. For each term t the inverted peer frequency is given by $IPF_t := \log(1 + \frac{N}{N_t})$, where N is the number of peers in the community, and N_t is the number of peers that offer documents containing term t.

In PlanetP summarizations of the content in the form of Bloom filters are used to decide what content a peer can offer. Thus, each peer can locally compute values for N and N_t. The relevance of a peer for answering multi keyword queries is then simply the sum of inverted peer frequencies for all query terms.

Peers are then queried in the sequence of their IPFs and the best documents are collected until queried peers do no longer improve the quality of the result set.

The authors show in [43] that in terms of retrieval effectiveness the approach is quite comparable to the use of inverted document frequencies regarding precision and recall. Also the overlap of documents retrieved using PlanetP is in average 70% compared to the use if inverted document frequencies (IDF). The summarizations are eagerly disseminated throughout the network by gossiping algorithms. Thus in terms of retrieval effectiveness this scheme describes documents on the summarization level, which is a suboptimal discriminator and by gossiping the system's scalability is limited. PlanetP needs continuous index updates when peers are joining or leaving the network, because a complete distributed index is maintained.

From the information retrieval systems point of view PlanetP brings GlOSS to a setting of moderately sized peer to peer networks by relicating the summaries. In a stable network, each peer knows the summaries of all other peers in the network.

3.2.2.2 Rumorama

The main drawback in PlanetP is the limited scalability caused by the summarization gossiping. In [89] Müller et al. propose first ideas how PlanetP could be made scalable. In the so called Rumorama network, each peer views the network as a small PlanetP network with connections to peers that see other small PlanetP networks; the small network viewed by a peer is called *leaf net*. One important aspect is that each peer can choose the size of the PlanetP network he wants to see according to its local processing power and bandwidth. Rumorama introduces hierarchies in PlanetP and thus aims to make it more scalable. In

Rumorama, each peer has two classes of neighbors: *Friends* and *Neighbors*. Peers do only know the IDs of its *Neighbors* and additionally content summarizations of its *Friends*. Since the amount data is made up from the summarizations, the costs depend on the number of *Friends* each peers has; the Rumorama protocol allows each peer to select the number of *Friends* independently from other peers (thus, the friendship relation is not symmetric).

This is done using a data structure called *Mask* (denoted as $mask(p_a)$ for a peer p_a), which is a prefix of the nodeID a peer has. A peer considers as friends all peers p_x that match $mask(p_a)$. Using masks, a peer can request from peers it knows summaries of peers. Thus it will receive and store only summaries matching the requested mask. The Rumorama protocol allows peers to chose the length of their mask individually. By choosing the proper length of the mask, a peer can effectively choose the number of summaries to be sent, received and stored.

3.2.2.3 Minerva

As PlanetP and Rumorama do, also the Minerva system relies on summarizations [26, 27]. The summarizations describe which information is available at each single peers in the network. The aim of Minerva is database selection in the context of peer to peer search.

While Rumorama replicates summaries, Minerva manages them in a conceptually global, but physicall distributed index that is layered on top of a Chord style distributed hash table, and closely follows a publish–subscribe paradigm. It holds a compact, aggregated meta information about the peers' local indexes. The extend of the meta information is defined by each peers limiting his willingness to disclose. It is assumed that every database forms a peer that is completely autonomous and has a local index.

The term space is partitioned into the distributed hash table, such that every peer is responsible for the meta information of a randomized subset of terms within the global directory; the lookup method of the distributed hash table is responsible for determining the peer responsibility for a particular term.

Every peer publishes a summary (*Post*) for every term in its local index to the underlying overlay network, which is routed to the peer currently responsible for this term. This peer maintains a PeerList of all postings for this term from across the network (for failure resilience and availability, the PeerLists may be replicated across multiple peers). *Post*s contain contact information about the peer who posted this summary together with statistics to calculate information retrieval measures (e.g. index statistics, or quality-of-service measures like average response times).

The querying process for a multi-term query proceeds as follows: first, the querying peer retrieves a list of potentially useful peers by issuing a PeerList request for each query term to the underlying overlay network. Using database selection methods, a number of promising peers for the complete query is computed from these PeerLists (peer selection). Subsequently, the query is forwarded to these peers and executed based on the their local indexes. Note that this communication is done in a pairwise point-to-point manner between the peers, allowing for efficient communication and limiting the load on the global directory. Finally, the results from the various peers are combined at the querying peer into a single result list (Result Merging) [27].

3.2.2.4 Adlib

Ganesan et al. propose Adlib [55], a system which aims to build a self tuning index which can dynamically trade off index maintenance cost against the benefits obtained for queries, in order to minimize the total cost if index maintenance and query execution [56]. Adlib introduces a two tier architecture [54], which partitions nodes into k independent equal sized *domains*, where k is tunable parameter controlled by Adlib. The nodes which are in one domain build a distributed index over the content stored in that domain.

Adlib distinguishes two different types of queries.

Partial lookup queries require a fixed number of results matching the specific key. Such a query is answered by first executing the query within the domain in which it is posed. If the number of answers is insufficient, the query is forwarded to more domains subsequently.

Total lookup queries shall return all available answers, and thus the query needs to be executed in all domains.

Self tuning. The goal of Adlib is an index that can dynamically trade off index maintenance cost against the benefits obtained for queries. Therefore, existing domains should be split into two when the current number of domains is less than the ideal value for the current network configuration. On the other hand, two domains should be merged into one when the number of domains is larger than the optimum.

3.2.2.5 PeerSearch

The idea of PeerSearch [116] (also called pSearch [117]) is to use CAN [99] in combination with the vector space model (VSM) and latent semantic indexing (LSI) [28] to create an index which is stored in CAN using the vector representations as coordinates. Latent semantic indexing.

PeerSearch represents documents and queries as vectors and measures the similarity between a query and a document as the cosine of the angle between their vector representations. This results in that indexes stored close to each other are also close in semantics. PeerSearch proposes to create a distributed index, which is partitioned placed on the network. This deterministic placement of content improves bandwidth efficiency by constraining the way a query is routed.

The components of PeerSearch are

P-VSM Content search using the vector space model. PeerSearch assumes a Zipf distribution of terms in a document, meaning that a small number of keywords can categorize a document's content. Each node is responsible for storing indexes containing some specific keywords. The vector space model is used to identify the important words in a document automatically. PeerSearch afterwards publishes this index of of the document to nodes responsible for those keywords. Processing a queries means to forward a query to nodes responsible for the keywords in the query; the nodes then search and return matching indexes using the vector space model.

P-LSI Semantic search using latent semantic indexing. Usually, CAN randomly generates document IDs and their coordinates are only used to store documents in the cartesian space. PeerSearch controls the placement of indexes such that indexes stored close to each other in CAN are also close in semantics. This is called a *semantic overlay*, in which routing in the distributed hash table is equivalent to searching in the semantic space. Using the semantic vector of a document as the key to store the document index in CAN achieves this goal. During retrieval, the semantic vector of the query is computed and query is routed in CAN using this vector as the DHT key.

PeerSearch works best in an environment where peers are reliably connected to the Internet. This is necessary because shared content is centralized in certain peers based on their hash values. The loss of a peer results in the loss of all its associated content or the transfer of massive quantities of data. Furthermore, it takes control of data placement out of the users' hands. For the global information used in the TFxIDF implementation statistics are assumed in PeerSearch; those statistics are precomputed using sample documents similar to those that will be used in the specific community.

3.2.2.6 Pier

PIER [64] is the acronym for peer to peer information exchange & retrieval. It combines flooding and distributed hast tables. The flood based network is used to find popular items, the distributed hash table is used to search rare items. The authors emphasize that one important scalability dimension is the degree of distribution. The goal is to build a query engine that scales up to thousands of participating nodes; this is discussed in contrast to the database community, where the largest database systems in the world scale up to at most a few hundred nodes.

However, in their presentation they show that they use DHTs as basic routing infrastructure, which forms as a tree, if all nodes route toward a single node. This provides a natural hierarchical distributed query processing infrastructure.

Pier uses the CAN realization of a distributed hash table. In their experiments, the authors also verified the use of Pier utilizing Chord as DHT. Applications use the PIER query processor to interact with PIER, which uses the DHT; on each node an instance of both PIER and the DHT are running.

3.2.2.7 PIRS

The major difference between PIRS [130] and other peer to peer information retrieval systems is that PIRS treats meta data as a dynamic resource that should be managed collectively by all peers. Yee et al. state that effective management of meta data improves query result quality; the better a file's body of meta data describes the file, the easier the file should be to find.

PIRS addresses the problem that users must know the exact meta data associated with a resource, e.g. the filename of a music file. The idea of PIRS ist, the each file is annotated with a set of meta data terms, contained in a descriptor.

Users create queries to find files in the peer to peer system. A query is a meta data set that a user thinks best describe a file. These queries are routed to reachable peers. (Queries generally do not reach all peers due to network conditions or details of the routing protocol.) Returned are pointers to instances of files that match the query, and the file's meta data set. The matching criterion is for all the query terms to be substrings of some term of the file's meta data set.

The authors emphasize the importance of dividing the information retrieval systems and the underlaying infrastructure. PIRS manages meta data in such a way as to gradually increase the variety of queries that can be answered for a given file. This is done by adapting the annotation of a particular file to match query patterns.

PIRS mainly has three components to achieve the mentioned goal.

Meta data collection is the process by which a file is annotated with identifying terms, which then are directly used for query matching. Items in the meta data body could be the name of a file or the type (e.g. PDF). The meta data elements are hashed in PIRS.

Meta data distribution is the process by which peers exchange meta data with each other in order to describe a file. During a query, it groups all meta data for each unique file in the results. When a user selects a file, meta data are heuristically replicated from the file's group onto the client.

Meta data use applies information retrieval techniques. The ranking function considered in PIRS are for instance term freuency and cosine similarity.

The most serious weak point in PIRS is that the authors ignore the specific problem which occur in peer to peer system when measures like the cosine similarity are used, i.e. they propose that a weighting as TFxIDF 'requires some modification, because global information on the number of documents in which each term appears, required by TFxIDF, does not exist in P2P systems'.

PIRS makes no assumptions about the underlying network, thus this authors propose that it can be easily combined with existing peer to peer infrastructures, i.e. Gnutella or FastTrack. Nevertheless, the evaluation done in the paper uses a simulation environment which is not discussed in relation to existing systems.

3.2.2.8 Other Systems

Some systems (of current research) don't already have a name, mostly are not that matured, or are not that closely related; but nevertheless they are well elaborated and thus worth mentioning in this context.

Zeinalipour-Yazti et al. present the 'Intelligent Search Mechanism' (ISM). It is based on an unstructured network. The basic idea is to exploit the locality of past queries. To achieve this, a peer estimates for each query, which of its peers are more likely to reply to this query, and propagates the query message to those peers only. Although they propose to use the cosine similarity for query evaluation, ISM just ignores the issue of collection wide information and computes scores based on local peer collections only [45]. The authors furthermore provide a simulation prototype which is called PeerWare that allows to validate various ideas and algorithms in a real setting [44].

Bawa et al. present SETS [25], a hybrid topology where the network is split into so called topic segments. These segments are characterized by their centroid description. SETS uses a vector space model to represent documents, peers and centroids. A query is first routed to the corresponding segment, and then evaluated using the segment subnet structure. An interesting evaluation result is that using the peer vectors for segmentation outperforms the usage of document vectors as basis for clustering.

Chunqiang Tang et al. propose eSearch, a peer to peer keyword search system based on a novel hybrid indexing structure [115]. In eSearch, each node is responsible for certain terms. It uses a distributed hash table (i.e. Chord) to map a term to a node where the inverted list for the term is stored. Terms central to a document are automatically identified by a heavy weight. In eSearch, only the term list for a document to nodes responsible for top (important) terms in that document is published; this optimization, however, may degrade the quality of search results: A query on a term that is not among the top terms of a document cannot find this document.

In REMINDIN' [29], a connection is scored by the similarity of the query topic to the topic(s) the target peer provides combined with a probability measure that the peer indeed will provide answers. To determine the similarity between query and content, both are annotated using concepts from a shared ontology.

Aberer et al. have build GridVine based on P-Grid. GridVine follows the principle of data independence by separating a logical layer, the semantic overlay for managing and mapping data and meta data schemas, from a physical layer consisting of a structured peer to peer overlay network; such separation is well-known from the database area and has largely contributed to the success of modern database systems. GridVine as the logical layer on top of the P-Grid physical layer handles the creation and indexing of RDF triples[8]. The meta data are spread using semantic gossiping, which aims at establishing global forms of agreement starting from a graph of purely local mappings among schemas. Peers that have annotated their data according to the same schema are said to belong to the same semantic neighborhood. Each peer has the possibility to create (either manually or automatically) a mapping between two schemas, in effect creating a link between two semantic neighborhoods. The network as such can be seen as a directed graph of translations [10]. GridVine supports RDQL query resolution through simple triple pattern combinations, following strategies similar to the ones presented in [35].

PeerDB [94] proposes a data model that standardizes the way data and services are published and queried. Thus, it focuses more on standards than on information retrieval itself. The three features of PeerDB are:

- Each peer is a object management system that supports content based search. In other words, PeerDB is a network of database enabled nodes.
- Users can share data without a shared global schema.
- PeerDB adopts mobile agents to assist in query processing, e.g. an agent may further manipulate the data retrieved from a node to ship back only summarized data, or filter away some objects.

PeerDB's peer to peer enabling technologies are provided by BestPeer [93].

[8] see http://www.w3.org/RDF/ for information about RDF

The open source project Edutella combines semantic web and peer to peer technologies in order to make distributed repositories possible and useful. Edutella deals with meta data about content, not with content itself. The Edutella network infrastructure builds upon the exchange of RDF meta data, with the query service as one of the core services of Edutella. The Datalog-based Edutella Common Data Model (ECDM) and the corresponding Edutella query exchange language (RDF-QEL) implements distributed queries over the Edutella network as well as distributed reasoning in this network. Routing is based on a HyperCuP based super peer backbone as well as additional indices to optimize routing. Edutella takes complex queries instead of keyword based search, distributes them to peers capable of answering the query, collects the answers and returns them to the originator.

Last but not least there are some frameworks like Odissea [114] which provides a distributed global indexing and query execution service. The authors suggest implementing any kind of ranking using their so called agnostic API (agnostic in the sense that it is not limited to a specific ranking algorithm). Odissea itself does not provide any information retrieval techniques.

3.2.3 Open Problems

The previous sections discussed state of the art systems in the area of peer to peer information retrieval. It can be summarized that they can be divided into two groups:

Distributed information retrieval system (e.g. PlanetP, PeerSearch) which all proactively create term (or peer) indexes, and second systems like e.g. PIER which strongly rely on a distributed hash table and thus do not allow for ranking of complex queries.

This implies that new algorithms should focus the two mentioned problems by avoiding the creation and continuous update of a global index and by allowing state of the art information retrieval ranking techniques for complex queries, including collection wide information.

CHAPTER IV

An Algorithm for Top-k Retrieval in Structured Peer to Peer Networks

The previous chapters introduced and discussed the aspects and research areas needed to develop a new algorithm for distributed top-k retrieval in peer to peer networks and presented the challenges for information retrieval in peer to peer network, see section 3.2.1. This chapter presents a new approach to retrieve the k best matching objects in a peer to peer network.

After motivating the concrete challenges for this algorithm, the single parts of the approach will be introduced. A mathematical analysis shows the correctness of the algorithm and the optimality regarding the delivered results. Finally, the basic top-k algorithm is extended for retrieval based on content and classification.

4.1 What is challenging?

As discussed before the new approach uses peer to peer, databases and information retrieval as background. All of the mentioned research areas has different challenges, whereof some are addressed in combination in this new approach for retrieving best matches in a distributed peer to peer network.

In section 3.1 the information retrieval background is discussed; indexes are one of the most important data structures used, e.g. using inverted file indexes for subsequent retrieval.

Maintaining these indexes, however, is a major problem in distributed systems, especially peer to peer networks that often share vast numbers of documents and have a high volatility with respect to peers joining and leaving the network. In contrast to static document collections every peer joining or leaving the network registers its document collection or removes it, thus indexes have to be updated very often.

For local query evaluation schemes a particular problem arises when collection wide information is an integral part of the query processing technique. As discussed in section 3.1.4 almost all all effective textual measures for information retrieval not only rely on statistics about the single documents, but also integrate statistics on the entire collection of all documents, e.g. how well individual keywords discriminate between documents with respect to the entire collection (inverted document frequencies). In the following TFxIDF is chosen for the discussion and included in the approach. Of course, any other technique that needs collection wide information can be integrated.

In the case of TFxIDF the term frequency may be locally evaluated for each specific document. However, for the document frequency a snapshot of the entire current content of all active peers needs to be evaluated. Of course this would immediately annihilate any benefits gained by sophisticated local querying schemes.

4.1.1 Example

A simple example shows how local scorings fail, if collection wide information has to be considered in the retrieval process. There are only two peers in the network and should return their best matches with respect to the most popular information retrieval measure TFxIDF. This measure is a combination of two parts, the term frequency (TF, measures how often a query term is contained in a certain document), and the inverted document frequency (IDF, the inverse of how often a query term occurs in the document collection). This measure needs to integrate collection wide information and cannot be determined locally.

The query Q is a simple conjunction of the terms 'retrieval' and 'peer' posed to the two peers P_1 and P_2; the query has to be evaluated locally at each peer, ignoring collection wide information. P_1 contains three documents D_1, D_2 and D_3. Peer P_2 also contains three documents D_4, D_5 and D_6. For simplicity of the example it is assumed that in D_1 to D_6 the two keywords occur mutually exclusive in the documents: D_1, D_2 and D_6 contain the keyword 'retrieval', whereas the documents D_3, D_4 and D_5 contain the keyword 'peer'.

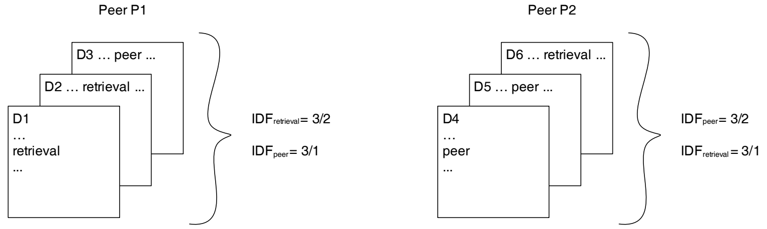


Figure 4.1: Collection wide information

Moreover, all documents are of the same length and the keywords occur in the same number in all documents, such that the respective term frequency is the same for all documents.

Evaluating the query Q locally means now to rank the documents in each peer. Since the keywords are mutually exclusive in the document base and the term frequencies are equal for each document, the ranking is only determined by the weighting factor of the occurring term in the inverse document frequency.

In P_1 there are two documents out of three containing the keyword 'retrieval'. This means an IDF of $\frac{3}{2} = 1.5$ for this term and only one document containing the keyword 'peer', i.e. an IDF of $\frac{3}{1} = 3$ for peer P_1. The result is that with respect to the query Q peer P_1 ranks the document D_3 better than D_1 and D_2.

Symmetrically, the peer P_2 ranks the document D_6 higher than D_4 and D_5, because here the keyword 'peer' occurs in two documents and term 'retrieval' only in one. Integrating

terms	TF_i					
	Peer 1			Peer 2		
	D_1	D_2	D_3	D_4	D_5	D_6
retrieval	$const.$	$const.$	0	0	0	$const.$
peer	0	0	$const.$	$const.$	$const.$	0

Table 4.1: Term frequencies

terms	IDF_i	
	Peer 1	Peer 2
retrieval	1.5	3
peer	3	1.5

Table 4.2: Inverted document frequencies

terms	TF_ixIDF_i					
	Peer 1			Peer 2		
	D_1	D_2	D_3	D_4	D_5	D_6
retrieval	$1.5 \cdot const.$	$1.5 \cdot const.$	0	0	0	$3 \cdot const.$
peer	0	0	$3 \cdot const.$	$1.5 \cdot const.$	$1.5 \cdot const.$	0

Table 4.3: TFxIDF values

the results from P_1 and P_2 gives a higher ranking for of D_3 and D_6 than of the four other documents.

In contrast, performing query Q over a central collection containing all six documents D_1 to D_6, one would find that both query term 'retrieval' and 'peer' occur in three of the six documents, i.e. have an IDF of $\frac{6}{3} = 2$. Since the term frequency is still the same, all six documents will be correctly assigned the same score.

4.1.2 Concrete Challenges

As shown, collection wide information is essential to provide proper document scores. But the index holding this information does not necessarily need to be completely up to date; obviously there is a trade-off between index information that is 'still current enough', given the network volatility and the accuracy of the query results.

Research on what dissemination level is required in Web information retrieval applications to allow for efficient retrieval showed that a complete dissemination with immediate updates is usually not necessary, even if new documents are included into the collection [125].

Moreover, the required level was found to be dependent on the document allocation throughout the network [124]: Random allocation calls for low dissemination, whereas higher dissemination is needed if documents are allocated based on content. Thus as lazy dissemination usually has comparable effectiveness as a centralized approach for general queries, but if only parts of the network containing most promising documents with similar content are queried, the collection wide information has to be disseminated and regularly updated.

As illustrated above, collection wide information is needed at each peer to do a correct score computation. Together with the peer to peer technology and the proposed information retrieval methods this implies concrete challenges:

Computation How to compute this information?

Storage Where to store it?

Distribution How to it distribute in the network?

4.2 Ingredients

The approach uses several ideas which are presented in the previous chapters (2.2.2.3.3, 3.1.4). This section correlates the aspects and shows how to put everything together.

The key to success is the observation that it is not needed to maintain a complete inverted index to process a query [90, 22]. In the example above, peers P_1 and P_2 need only IDFs for terms a and b, but not for all terms occurring in their documents to calculate the correct scores.

In the previous section the concrete challenges for the algorithm were discussed. In the following, the building blocks of the approach are introduced and put together to a self-contained algorithm.

4.2.1 Overview

This section briefly gives a high level overview what the building blocks are and how they are used to address the mentioned challenges.

4.2.1.1 Storage, Distribution, and Computation

Before the elements of the algorithm are presented, this section briefly introduces how the mentioned challenges are faced and which solutions ProToRaDo provide for each aspect.

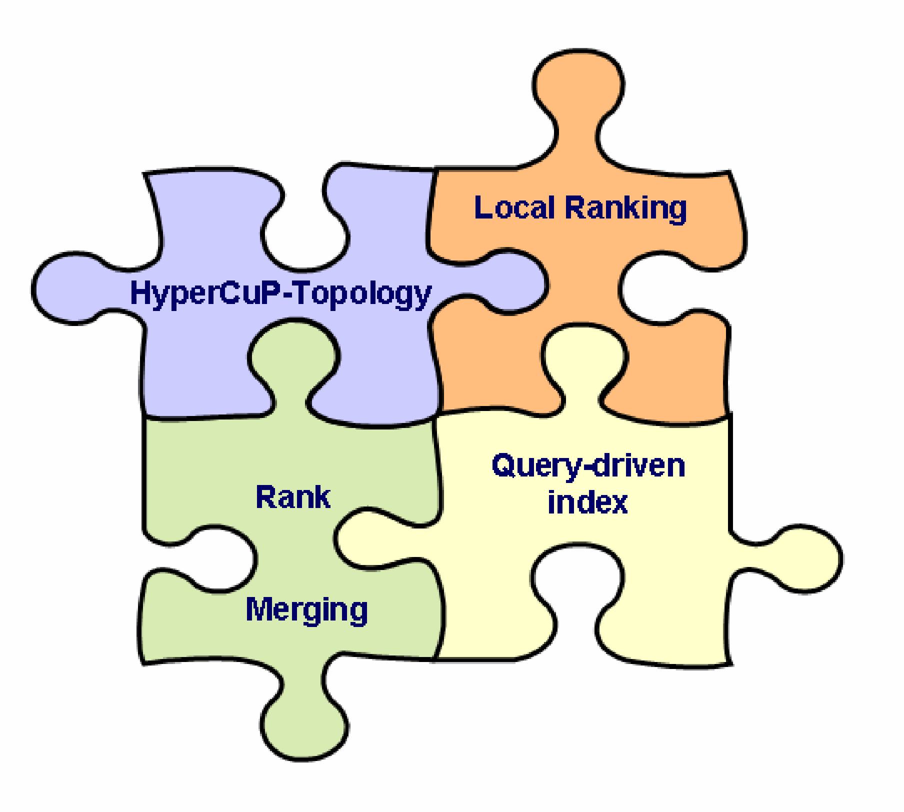


Figure 4.2: Ingredients of ProToRaDo

4.2.1.1.1 Storage To store this information, a super peer network approach is used [129]. Here, the index management responsibility is assigned to the super peers. The super peers form the network backbone, and each document provider peer is directly connected to one of them. The backbone of super peers arranged in the HyperCuP topology in used for two reasons:

1. Updates in DHTs are very expensive, s. section 2.2.2.3.
2. Collection wide information is needed. This means one has to contact all the peers. This cannot be done using a topology which only allows lookups like the distributed hash tables. Thus, broadcasting is needed in a very efficient way. As discussed in section 2.2.2.3.3 using the super peers arranged in the HyperCuP topology is a very good solution.

4.2.1.1.2 Distribution A simple query consists of a conjunction of keywords that are searched in the content of the documents. Later, the extension for categories will be introduced, which leads to an additional query element: a category of the taxonomy which should be searched.

Before distributing such a query, a super peer adds the necessary collection wide information from its index to it. If it isn't yet in the index, an estimation is provided.

The category in the query is used as a filter for two purposes: First to reduce the number of peers (and thus documents) which must be searched and ranked. Second the user can use the category to avoid ambiguities. If a user is interested in the local sport-team called 'Jaguars', the appropriate category will avoid hits Jaguar-cars and the animal Jaguar. The keywords which are specified in the query will only be used for the documents which are in the named category.

4.2.1.1.3 Computation Responding peers do not only deliver matching documents, but also each add local data necessary to compute the collection wide information (for TFxIDF this is the document frequency for each query term and the document count). On the way back to the originating super peer, this data is aggregated. Thus, the originating super peer gets everything it needs to compute the complete aggregate, and can store the computed result in its index.

4.2.1.2 Basic Strategy

After presenting the building blocks, this section describes the basic application flow.

The top-k answering and routing algorithm is based on local rankings at each peer, aggregated during the routing of answer for a given query at the super peers involved in the query answering process.

Each peer computes local rankings for a given query, results are merged and ranked again at the super peers and routed back to the query originator. On this way back, each involved super peer again merges results from local peers and from neighboring super peers and forwards only the best results, until the aggregated top-k results reach the peer that issued the corresponding query.

While results are routed through the super peers statistics which peers / super peers returned the best results are maintained. This information is subsequently used to directly route queries that were answered before mainly to those peers able to provide top answers.

One important aspect is the volatility of a peer to peer network. Peers join and leave arbitrarily and/or the documents at the peers change (adding, deleting, editing).

The typical approach used here is a time-stamp, which is the well known from the area of caching. Each index entry gets a time to live, or better: is assigned a best before time-stamp. Every time an index entry should be used it is checked whether the times-tamp is too old or not. If yes, the query is broadcast and thus, the indexes are updated. If not, the index information is used for optimized routing of the query to the peer which previously contributed to the best results.

This approach was implemented and works very well. Of course, one can think of alternatives. One possible way is to send a query to peers randomly, not only to the peers which contributed to the top-k previously. Such random forwarding means that queries are routed using the information in index. For a previously posed query there is an index entry which allows to route only to the peers which did contribute to the result set previously. To take volatility into account the query is then also send to a small number of peers randomly. Since the time-stamp works fine, ideas for alternatives are designated further work.

4.3 Algorithm Details

In this section the data structures used in ProToRaDo are introduced. For simplicity only the structures are discussed and it is assumed that there are accessors and methods for e.g. adding or deleting items of sets as accepted for object oriented programming. The structures and methods of the algorithm were implemented in Java; thus here pseudo code is presented to offer a more abstract view of the algorithm.

4.3.1 Top-k Retrieval Algorithm

This section presents ProToRaDo, an algorithm for basic top-k querying capabilities in peer to peer networks with minimum transfer of object data. According to the distributed nature of the retrieval and the peer to peer network, the distributed retrieval algorithm is divided into three parts that are respectively executed by

- the super peer initially receiving the query,
- the super peers in the HyperCuP backbone, and
- and the local peers at each super peer.

Since a dissemination of global knowledge should be avoided due to the overhead of data transmission, a basic concept of the algorithm is to locally evaluate as many parts of the query as possible. This means only the super peer receiving the query (i.e. the root node of our implicit HyperCuP spanning tree) needs full information to control the execution of the queries in order to guarantee a correct top-k result set with a minimum transmission of data. This super peer will hand on the query to the relevant super peers along the backbone of adjacent super peers, which in turn will forward the query to their relevant adjacent super peers and connected local peers, without having to have full information about how the query answering is progressing.

The local peers will just execute the query over their local object collections or databases and retrieve some best matching objects. All relevant steps are presented in detail in the following.

Assumptions. ProToRaDo relies on a set of super peers managing a number of local peers and interconnected by a backbone using the HyperCuP topology. It is also assumed that all peers are cooperative and provide normalized scores that can be compared to distinguish the quality between different objects.

Each super peer SP manages an index I_{SP} in that information about which of its local peers and adjacent super peers contributed results for answering recently posed queries like shown in. These indexes can be maintained efficiently even in rather volatile peer to peer networks to hold 'current enough' information about object distributions. All index entries are time stamped and expire after a certain time, which is set depending on the volatility of the network. Thus the individual index entries can be kept 'up-to-date enough' to allow for improved query processing even in volatile networks.

Now the algorithm to answer a top-k query Q posed by peer P to super peer SP is presented. The algorithm is entirely controlled by super peer SP, which - whenever necessary - poses requests to connected peers and super peers for localized information gathering.

Since all super peers are organized in a HyperCuP topology and SP is the root node of an implicit spanning tree containing all super peers, the notion of *adjacent* super peers of SP is used, i.e. those super peers that can directly receive messages from SP, but are more distant from the root node, i.e. whose HyperCuP edge label is smaller.

Moreover, it is assumed that the query is answered using a snapshot of the current peer to peer network at query time, i.e. the connections stay constant for the time of the query answering process. Disconnections during query processing will be discussed later in the discussion of the transaction handling by introducing time outs for peers that do not answer a request within a tolerable time span.

ProToRaDo-Algorithm

0. Assign a unique transaction identifier T depending on the query Q, querying peer P and super peer SP. Initialize a counter $i := \emptyset$.

1. Choose the participating peers and super peers for answering the query Q:
   If query Q is contained in index I_{SP}, and this index entry is not expired, assign sets of contributing local peers P_T and adjacent super peers SP_T as given by I_{SP}, else set P_T as the set of all locally connected peers and SP_T as the set of all adjacent super

peers.

2. Initialize a data structure $TopRes_T$ as a $|P_T| + |SP_T|$ – dimensional array of o_{id} and *score* pairs. Assign each (super) peer in P_T and SP_T to a specific field in $TopRes_T$. Initialize three sets $BestPeers_T := \emptyset$, $Delivered_T := \emptyset$, and $RequestResults_T := \emptyset$.

3. Send an $open\_transaction(T, Q, SP)$ request to each (super) peer in P_T and SP_T and add the respective (super) peer to set $RequestResults_T$.

4. Send an $get\_next(T, Q, SP)$ request to each (super) peer in $RequestResults_T$ and remove the respective peer from $RequestResults_T$.

5. For each incoming message that a (super)peer cannot deliver more result objects, send a $close\_transaction(T, Q, SP)$ request to the (super) peer, remove the (super) peer from P_T or SP_T and delete its assigned field in $TopRes_T$.

6. For each incoming ($o_{id}/score$) pair from (super) peers with respect to transaction T do

    6.1 If $o_{id} \notin Delivered_T$, store the ($o_{id}/score$) pair in the respective field in $TopRes_T$ assigned to the delivering (super) peer, else discard the pair and add the delivering (super) peer to $RequestResults_T$.

7. If there are still missing entries in any field in $TopRes_T$, proceed with step 4.

8. Select all distinct objects with current maximum score from $TopRes_T$. While $i \leq k$ and there are still objects, do:

    8.1 Pick any object o's o_{id} having maximum score and deliver its o_{id} and *score* to peer P as the i^{th} best object.

    8.2 Add object o's o_{id} to the set $Delivered_T$ and increase $i := i + 1$.

    8.3 Remove o's o_{id} and *score* from all occurrences in $TopRes_T$ and add the corresponding (super) peers of the respective fields to $BestPeers_T$ and $RequestResults_T$.

9. if $i > k$, send $close\_transaction(T, Q, SP)$ request to all (super) peers in P_T and SP_T and update I_{SP} for query Q using the (super) peers in $BestPeers_T$. Discard all temporary results and data structures and terminate the algorithm.

10. If set $RequestResults_T$ is empty, abort query Q at peer P with the message that only i results are available and discard all temporary results and data structures, else proceed with step 4.

Please note that although generally speaking the number k of objects to be returned is an integral part of the query Q, the sets P_T and SP_T in step 1.1 can also be assigned, if index

I_{SP} does not contain query Q, but query Q^* with the same query predicates, but a larger number of objects to return than k. The resulting sets P_T and SP_T will in that case not be optimal for query Q, but usually still result in much better performance than simply flooding the query through the network.

Another interesting side effect is the successive output of result objects in step 8.1 such that the user can already investigate some first overall best result objects before all k overall best result objects have been found. Though the total retrieval time stays the same as in the case where all objects are returned after all top k objects have been determined, the psychological response time is reduced for the users. Moreover, once a user is satisfied by the object(s) from the result set retrieved so far, she can terminate the query at an early stage before the full result set has been retrieved and thus improve bandwidth usage.

Next, the functions that are called in our distributed algorithm requesting locally connected peers and adjacent super peers to join into a query processing task and to deliver their best matching objects, are considered. These function calls are $open\_transaction(T, Q, SP)$, $get\_next(T, Q, SP)$, and $close\_transaction(T, Q, SP)$. Since their basic functionality does not differ for peers and super peers, though their local execution has to be slightly different, it is assumed that their individual implementations are simply overwritten to suit the respective (super) peers. For the requesting super peer SP their purpose, interface, and expected results do not differ between peers and super peers. Starting with the implementations of the functions in local peers, afterwards the functions in the super peers handling the local aggregation tasks will be discussed.

Since local peers may differ in their information management and querying techniques, their implementation may essentially differ between peers. Some peers may rely on a database management system to store and query data, while other may use a variety of custom made applications to manage their data. It is assumed a component in each peer that wraps the results and messages according to the super peers' needs, and leave the actual local top-k querying and scoring to the individual peers. For example peers relying on a local DBMS may use an algorithm like [61] whereas other peers may rely on filtering techniques for their collections. In the following it is only assumed that if asked to, every peer joins a transaction, is able to evaluate a top-k query locally, iterate over the respective result set, and deliver its objects using a specific o_{id} (e.g. an URI, etc.) together with a distinctive score value normalized to the interval $[0, 1]$. When running out of deliverable objects a peer notifies its super peer with a suitable message. The requests sent to a local peer are:

Basic functions at each local peer:

open_transaction(T, Q, SP):
If a peer receives this request it will prepare to answer the top-k query Q over its local data (e.g. open a result set and position a cursor) and assign all further requests with the identifier T to the respective result set.

get_next(T, Q, SP):
If a peer receives this request it will iterate over the result set assigned to T (e.g. move the cursor or get the next best object from a progressive retrieval algorithm) and send the respective result object's o_{id} and *score* to super peer SP. If there are no more results that can be delivered, if will send a respective message to super peer SP.

close_transaction(T, Q, SP):
If a peer receives this request it aborts the query assigned to T (e.g. close an open result set) and may discard the related temporary results.

The functionality in super peers is a bit more difficult than for the local peers, but quite similar to respective steps in the algorithm at the querying super peer. Assume that a super peer SP sends a request to an adjacent super peer SP^*. To open the transaction for a specific query this super peer SP^* will also have to choose local peers and his adjacent super peers to participate in the query from its local routing index. If requested, it will have to aggregate information, determine the current best object locally, and hand it on to the super peer that requested it. Eventually the super peer will have to close the transaction:

Functions at each super peer SP^*:
open_transaction(T, Q, SP):

1. Choose the participating peers and super peers for answering the query Q: If a query Q posed by SP is already contained in index I_{SP}, and this index entry is not expired, assign sets of contributing local peers P_T and adjacent super peers SP_T as given by I_{SP}, else set P_T as the set of all locally connected peers and SP_T as the set of all adjacent super peers.

2. Initialize a data structure $TopRes_T$ as a $|PT| + |SP_T|$ – dimensional array of o_{id} and score pairs. Assign each (super) peer in P_T and SP_T to a specific field in $TopRes_T$. Initialize three sets $BestPeers_T := \emptyset$, $Delivered_T := \emptyset$, and $RequestResults_T := \emptyset$.

3. Send an *open_transaction*(T, Q, SP) request to each (super) peer in P_T and SP_T and add the respective (super) peer to $RequestResults_T$.

get_next(T, Q, SP):

1. If set $RequestResults_T$ is empty, send a message to super peer SP that no more objects can be delivered, else do

(a) Send a $get\_next(T, Q, SP^*)$ request to each (super) peer in $RequestResults_T$ and remove the respective peer from $RequestResults_T$.

(b) For each incoming message that a (super) peer cannot deliver more result objects, send a $close\_transaction(T, Q, SP)$ request to the (super) peer, remove the (super) peer from P_T or SP_T and delete its assigned field in $TopRes_T$.

(c) For each incoming $(o_{id}/score)$ pair from (super) peers with respect to transaction T do

i. If $o_{id} \notin Delivered_T$, store the $(o_{id}/score)$ pair in the respective field in $TopRes_T$ assigned to the delivering (super) peer, else discard the pair and add the delivering (super) peer to $RequestResults_T$.

(d) If there are still missing entries in any field in $TopRes_T$, proceed with step 1b.

(e) If there are objects in $TopRes_T$, select an object with current maximum score from $TopRes_T$. and deliver its o_{id} and score to super peer SP. Add the object's o_{id} to the set $Delivered_T$ and remove the object's o_{id} and score from all occurrences in $TopRes_T$ and add the corresponding (super) peers of the respective fields to $BestPeers_T$ and $RequestResults_T$.

$close\_transaction(T, Q, SP)$:

1. Send a $close\_transaction(T, Q, SP)$ request to all (super) peers in P_T and SP_T and update I_{SP} for query Q and querying peer SP using the (super) peers in $BestPeers_T$.
2. Discard all current sets and data structures.

Please note that for using the index I_{SP^*} in processing the $open\_transaction(T, Q, SP)$ request, it is of essential importance that the query Q has been posed by super peer SP before, as the child nodes (and thus adjacent super peers that can deliver relevant results) depend on the respective edge weight of the implicit super peer spanning tree in the HyperCuP topology. Thus all the knowledge of which adjacent nodes may provide interesting information for the top-k case, is dependent on the topology for each query instance. Due to the characteristics of the HyperCuP topology it is irrelevant, whether SP is the querying peer running the top-k search, or just passing the query on. It is only important that the query was routed via the edge between SP and SP^*. As stated before, index entries for identical queries delivering more top elements as in the current request, can be used instead of querying all peers and adjacent super peers.

The index I_{SP} always holds information for routing optimization in sense of avoiding irrelevant destinations. For each query it contains the peers, which have recently contributed to a top-k result set. While results are delivered by the super peers, for each query transaction T one maintains statistics in a set $BestPeers_T$, which peers/super peers returned the best results. On query completion this information is used to update all local routing indexes. To adapt to changes in the peer to peer network, all index entries expire after a specified time span. The more volatile the network is the shorter the expiration period has to be in order to adapt to changing data allocations. Query driven update of indexes i possible, because queries are not posed randomly, but usually follow a Zipf distribution, where few queries make up the majority of all requests. Zipf distributions are ubiquitous in content networks, the Internet and other collections and have become one of the most empirically validated laws in the domain of linguistic quantities and networks (in the form of power law distributions) [14].

Example. To show in more detail how the algorithm works, an an example for the simple scenario given in figure 4.3 is considered. Given is a backbone of four super peers each accommodating two local peers. A top-k query Q by peer P_Q will define SP_A as root of the super peer backbone spanning tree (cf. labeled edges in figure 4.3) and start the query processing in SP_A for transaction T. Assume that the query is a top 2 query that has recently been posed. SP_A can check its local index to find out that only P_1 and SP_B have contributed to the result (step 1.1), i.e. P_T and SP_T are single element sets containing P_1 and SP_B respectively. Assuming that the peer to peer network is not too volatile, there won't present index entry expiration in this example and initialize a two-dimensional array containing the current top elements of SP_B and P_1 (step 2). Note that in contrast to flooding query frameworks SP_C will not be bothered at all with query answering, since SP_A already knows that it most probably cannot deliver suitable results. SP_A then will open transactions in P_1 and SP_B (step 3) and add both to a set $RequestResults_T$.

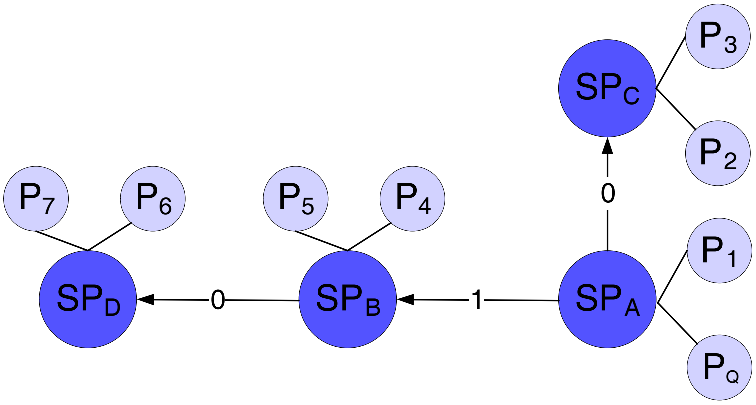


Figure 4.3: Querying a super peer based peer to peer network

Receiving the request P_1 will perform its local query, whereas SP_B will look up its index and may find that the query was recently posed by super peer SP_A and the top 2 results came from SP_D and P_4. Subsequently SP_B will open transaction T in P_4 and SP_D (which may then in turn open the transaction in say P_7) and initialize all data structures needed. Now SP_A will request the top objects from P_1 and SP_B (step 4). Assume that P_1 offers an object o_1 with score 0.8. Receiving the request SP_B will in turn request the top results from P_4 and SP_C, which in turn will request the top object from P_7. SP_B has to wait until it has both results from P_4 and SP_C (to be on the safe side there will be a timeout after which SP_B will assume that any (super) peer, which has not yet delivered, has dropped out and thus will not be considered in the query any further). Assume P_7 delivers an object o_2 with score 0.7, which is handed on by SP_C to SP_B. SP_B also receives the top object o_3 with score 0.9 by P_4. SP_B now has to pass on the maximum object o_3 to SP_A, which in turn chooses the maximum object o_3 (step 8) and passes it on to P_Q as the overall top object. Since the top 2 objects are needed and the best object of P_1 is still valid, SP_A will request the next object only from SP_B who in turn will only request the next object from P_4 (say o_4 with score 0.7) and deliver it to SP_A (steps 4-7). Now SP_A can again choose the top object o_1 by P_1, deliver it to P_Q (step 8) and close the transaction (step 9). Since the top objects came from SP_B and P_1 there is no need to update $SP_A's$ index. SP_B will in turn close the transaction in P_4 and SP_C, however remove SP_C from its local index, because it did not contribute to the result of query Q.

4.3.2 Correctness and Optimality Results

The discussion of correctness of retrieval results in peer to peer networks is a difficult matter because of their volatile nature. In traditional database retrieval complex transaction protocols have been designed to assure that no phantom objects occur in the retrieval process (e.g. when objects are updated) and it is easy to decide, if a retrieval algorithm has left out relevant results or retrieved false results (cf. precision/recall in IR). In exact match peer to peer retrieval each peer returns all its results matching the query (flooding of queries) or the objects are retrieved after a matching entry (and thus the address of a peer offering a result object) in some centralized or distributed index structure has been found. However, it cannot be guaranteed that

- a peer is always online for the necessary time to transfer these result objects,
- a peer for a matching index entry is still available,
- or that no new peer offering relevant content has occurred after a part of a distributed index has been evaluated.

Thus the correctness of retrieval results in peer to peer retrieval is always considered on a ‘static snapshot’ of the peer to peer network. If no peers drop out or new peers are registered between the atomic evaluation of the query and the delivery of the result set, the query has been answered correctly. Such a model becomes of course less adequate the more volatile a peer to peer network gets.

The problem how to define correct retrieval gets even worse, if top-k queries have to be answered. Besides the difficulties with the volatility of the peer to peer network, also the heterogeneity of the peers plays an important part, since each peer only knows its local objects and different peers may also feature different scoring or retrieval strategies. To compare between objects is thus only possible, if the cooperative behavior of all peers is assumed and a normalization of the score values (e.g. to the interval $[0, 1]$) is requested of each single peer. Nevertheless different peers may still score the same object differently. Since the only solution of retrieving all objects and scoring them centrally with a single method (even the naïve approach of retrieving only the top k objects of every peer and then rescoring results does not solve this problem, because different scoring methods usually do not maintain relative monotonicity) is clearly impractical and generally not desirable in the peer to peer context anyway, always use the maximum score value any queried peer has assigned to an object will be used as the relevant score for this object.

Since [92] shows that querying with partial indexes along a super peer backbone (pointing always to the top peers, which have recently contributed to the retrieval result of the same query) usually results in a good enough response behavior, this algorithm will combat the problems of volatility and heterogeneity relying on the experiences with such indexes. Therefore it is investigated correctness regarding correct top results with respect to the relevant part of a minimum spanning tree induced by an index entry maintained by the super peer receiving the query. Moreover, it is assumed that the additional querying of arbitrary (super) peers in step 1.2 will be sufficient to react to changes of content allocation within the network.

Lemma 1 (Correctness of the Top-k Result Set):
Given that only cooperative peers are part of the peer to peer network and retrieval scores between peers can be compared with respect to the objects' relevance to answering the top-k query, i.e.

- individually assigned scores are reliable and no peer maliciously assigns high scores to irrelevant objects
- for any two score assignments for different objects from different peers, it can be correctly decided, whether one of the objects is better than the other(s),

the distributed top-k algorithm always retrieves the correct set of k overall best objects with respect to the index of the querying super peer, if the maximum assigned score is used as the relevant object's score in the case of some peers assessing the score for a database object differently.

Proof:

Since – as discussed above – the correctness of a result set can only be decided at a certain point in time over a fixed set of database objects/documents, it is considered the retrieval situation for the algorithm with up-to-date indexes in the super peers and it is assumed that neither new documents are added to the collection, nor are documents deleted. For the proof an inductive argument based on the aggregation of the local rankings is used. Since the indexes are up-to-date it is known that some overall top scored object o_{top} is in one of the local collections of at least one peer in some index. Without loss of generality it can be assumed there would only be a single object o_{top}. For the sake of contradiction now it is assumed that the querying super peer would deliver a dominated object o_x with score(o_x) < score(o_{top}) as overall top-scored object. Since all super peers indexes are up-to-date and it is known from step 1 that all relevant peers have been queried and their best objects have been requested (step 4), there must be a path through the peer to peer network from the querying super peer to at least one peer holding o_{top}. This local peer has to return o_{top} as best object to its connected super peer due to the monotonic iteration through scorings in each local peer (i.e. at any point in time every local peer on request returns the highest scored object it can offer and which has not yet been output). Otherwise, as always the overall maximum scoring are used as relevant score for each object, there would exist an object having a strictly better score than o_{top}. Moreover, since there is a path along the super peer backbone and each super peer has to wait until all results have arrived (step 4-7) before determining the local maximum score within its local collection (step 8) all super peers along the path from the querying super peer to the local peer hosting o_{top} also have to select o_{top} as best object and pass it on, until it eventually arrives at the querying super peer. Thus by delivering ox the super peer would have chosen a dominated object though knowing o_{top}, which is a contradiction to the maximum search in step 8.1 of the algorithm. Inductively and considering that already delivered objects offered by any local peer are replaced by the next best objects (i.e. with monotonically decreasing score values) this peer can offer (step 6.1), the correctness of the best k retrieval results is guaranteed.■

Having proven that a correct result set with respect to an up-to-date index is retrieved by our algorithm, one also has to consider the overall costs of transmitting the necessary information about object data. The next lemma shows that in order to retrieve the result set, only a minimum amount of object data needs to be transmitted.

Lemma 2 (Optimality of transferred object data):

Given the assumptions and conditions of lemma 1, the distributed top-k retrieval algorithm needs to transfer only a minimum amount of object data to find the correct result set.

Proof:

To show the optimality of the transmitted object data the focus is on the score constraints that are maintained by the local peers and super peers along the backbone. To decide for the correct maximum in each single super peer one has to request the best objects from each local peer in its index and the adjacent super peers towards the leaves of the minimum spanning tree. Once the maximum is chosen, the best available objects are still correct for all but one local peer or adjacent super peer. For the next maximum choice only the next best object has to be requested from this and only this (super) peer (unless the same maximum object was offered by two or more peers in which case step 8.3 will also request next best objects from these other peers). As the querying super peer at the root of the super peer backbone knows what object was returned as part of the retrieval result, which (super)peer's best objects do still hold and how many objects are still needed, the iterative requests of next best objects have to be issued by this super peer as in step 4. Therefore only (super) peers have to refresh their offer (and thus transmit more object data), whose current best offers have already been part of the final result set. This transmission is necessary, since after delivery of a peer's best object, the next best object of this peer can still have a higher score than the best objects of the other peers at the same super peer. Omitting the transmission when starting a new search for the maximum scored object in a super peer could violate the correctness of the result set and hence the amount of object information transmitted is optimal. ■

The last lemma is useful for handling disappearing peers in volatile networks. The typical situation in top-k queries is that a user poses a query on a rather small amount of objects/documents to retrieve. Typical values for k can be assumed to range around 10 in most practical applications. The result set delivered can then either consist of k pointers to the peers and objects/documents together with the actual score value or the actual objects/documents themselves together with the respective scores. Given that the network can be volatile (especially since big parts of the objects under consideration have to be handed on along the super peer backbones and each super peer has to wait until it has collected all necessary objects), it can happen that a peer vanishes after one of its objects has been delivered as part of the result set. If the original document has been delivered in the result set, this does not pose a problem. In contrast, if only a pointer has been delivered, the retrieval of the result document fails and there will be less than k objects. So immediately delivering the objects/documents often is a sensible approach that – given the result of lemma 2 – does not result in unnecessary waste of valuable network bandwidth. In some situations, applications might favor optimizing the number of messages instead of bandwidth, e.g. if the bandwidth is large, but the RTT long. In this case our algorithm can be straightforwardly adapted by letting *get_next*() return a set of up to k results instead of just one result per message. The optimal value for this batch transfer could even be chosen depending on the actual connection characteristics between two nodes.

4.3.3 Extending to Content and Classification

So far the described algorithm works based on the content of the documents in a collection. Today, information are more and more classified. Taxonomies and ontologies help to categorize documents.

Therefore, ProToRaDo was extended to take into account that users might want to search not only giving some keywords, but want to restrict the search to a particular topic in a hierarchy.

4.3.3.1 Example: News Corpora in Digital Libraries

Digital libraries today offer a wide variety of content that is usually accessible through central portals. Though this generally is a useful way to access individual sources, it hampers the creation of federated networks of libraries or document collections. But such federations would be especially valuable for finding documents best matching the user's information needs and possibly providing a number of different views or opinions on a topic. One possibility of facilitating federated searches are meta-crawlers. But documents only accessible via a certain portal interface often cannot be crawled, also known as the 'hidden Web problem'. Moreover, crawlers only periodically crawl collections and update their indexes, and thus cannot react flexible to new, previously unknown collections and content changes in existing collections.

Nearly every (digital) library offers access to news articles, thus the motivation here is the example of federated news collections. News items can also be compound documents and are usually categorized within certain topics like politics or sports. Periodic changes of user interests are assumed, which also allows also to experiment with the arrival or removal of complete corpora from our federation.

4.3.3.2 Query Processing

Since the basic algorithm discussed above does not change, in the following only the additional constructs are introduced.

4.3.3.2.1 Query distribution at super peer Each super peer maintains an IDF index containing IDF values for the keywords. This is done separately for each category, not for all documents in all categories. Thus, the key for this index is built from a category and one keyword. As mentioned above a query contains the IDF values for the query-terms.

The IDFs are taken from the IDF index, or estimated if a keyword is not yet in the index. In the latter case, the average IDF is used as estimation.

4.3.3.2.2 Query Processing at Peer At each peer, first only documents in the specified category are taken into account. The top-k documents are determined using the TFxIDF algorithm, but based on the IDF values from the query. If this gets enough ($=k$) results in the queried category, these are sent to the super peer. If the number of documents matching the query is smaller than k, the query is relaxed, first to subcategories and then to the super-categories. This process is repeated until the peer has k results or the root of the taxonomy is reached. The entries found in all newly searched categories are sorted by their similarity to the originating category using the following measure (from Li et al. [75]):

$$sim(c_1, c_2) \quad = \quad e^{-\alpha l} \quad \cdot \quad \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}}$$

where l is the shortest path between the topics in the taxonomy tree and h is the depth level of the direct common subsumer. α and β are parameters to optimize the similarity measurement (best setting is usually $\alpha = 0.2$ and $\beta = 0.6$).

The super peer then gets the top-k of the peer or a number $n < k$ of documents matching. This query relaxation is shown in algorithm 2.

4.3.3.2.3 Result merging at super peer A super peer retrieves max. k hits from each of its peers and combines them to the top-k. As described above it is possible that peers also send results which are coming from another category as requested. In this case, the super peer first takes all hits which match the queried category. If this results in a set smaller than k it takes the next best-matching hits from each peer and combines them using the described sorting.

Algorithm 2 Query relaxation

```
ResultSet results $\leftarrow$ new ResultSet();
searchRoot $\leftarrow$ Query category;
**repeat**
  Initialize a new set searchCategories
  add searchRoot to searchCategories;
  **while** number of results $< k \wedge$ searchCategories $<> \emptyset$ **do**
    Initialize new set allChildren
    **for** all $cat \in$ searchCategories **do**
      Initialize ordered list matchingDocuments {retrieve hits matching category exact}
      mdocs $\leftarrow$ mdocs $\cup$ documents;
      sort(mdocs, compare);
      mdocs $\leftarrow$ topKof(mdocs);
      allChildren.add(cat.children)
      searchCategories $\leftarrow$ allChildren; {go one level down in category tree}
      removesearchRoot from searchCategories; {do not to traverse subtree twice}
    **end for**
  **end while**
  searchRoot $\leftarrow$ parent of searchRoot {go one level up in category tree}
**until** numberOfResults $\geq$ k $\vee$ searchRoot = topOfTaxonomy
**return** topKof(rs);
```

Algorithm 3 compare(value$_1$, value$_2$)

```
**Ensure:** The higher valued document is returned based
  on TFxIDF-score and category-similarity
value$_1$ > value$_2$ $\Leftrightarrow$ score$_{cat}$(value$_1$) > score$_{cat}$(value$_2$)
                     $\vee$ score$_{term}$(value$_1$) > score$_{term}$(value$_2$)
```

4.3.3.2.4 IDF index update The IDF index can be updated in two ways:

1. By summing up document frequencies and document counts delivered from connected peers and super peers, the super peer where the query originated computes IDFs for each query term and updates its IDF index. If the difference between computed IDF and estimated IDF value exceeds a threshold, the query is redistributed, this time using the computed IDF values.
2. if a super peer receives a query it checks, if the IDFs contained are marked as estimated. If this is not the case, these values are used to update the IDF index.

4.3.4 IDF Index Entry Expiration

Viles and French have shown that in a large document collection IDF values change slowly [125, 124] In our context, this is not strictly applicable, because there are two kinds of changes that may influence our collection wide information significantly:

1. *New documents with similar content: new peers join the network.*
   Imagine a large federation of library servers which offer articles from different newspapers. Let's assume there is already a newspaper like the NY Times in the collection. What can happen if peers join the network offering a new newspaper, i.e. the LA Times? In this case one can be sure that the articles usually will be on nearly similar topics except a few local news. Thus, one does not really have to update our IDFs since the words in the articles are distributed the same way as before.
2. *New documents or new corpora: New library servers join the federation or new documents are included in existing collections, whose content is very different from existing articles and thus shifts IDFs and changes the discriminators.*
   Let's look at an example: Assume there is an election e.g. in France and people use our P2P-news-network to search for news regarding this election. This normally will be done using queries like ‘election France’ and results in a list of news that contain these words. In this case there would be a lot of news containing France, thus ’election’ is the discriminator, and the IDFs will give us the correct results. Now think of another election taking place in the US in parallel. The term ’election’ will no longer be the best discriminator, but the term ’France’ then gets more important.

In these cases one has to solve the problem that entries in the IDF index become outdated over time. Both cases can be handled in the same way: Each IDF value gets a timestamp when the term appears for the first time and the term/IDF-pair is stored. After a specific expiration period (depending on the network-volatility) the item becomes invalid and the entry gets deleted. In this way the IDF recomputation is forced if the term occurs again in a query. By adjusting the expiration period one can trade off accuracy against performance. The expiration period for terms occurring more frequently is reduced, thus ensuring higher accuracy for more popular queries.

4.3.5 Query Routing Indexes

So far, still all queries are distributed to all peers. Broadcasting can be avoided by introducing additional routing indices which are used as destination filters:

- For each category in our taxonomy the *category index* contains a set of all peers which have documents for this category. It is not relevant if this peers did contribute to queries in the past.
- In the *query index* for each posed query the set of those peers which contributed to the top-k for the query are stored.

4.3.5.0.5 Query Distribution The super peer first checks if all query terms are in the IDF index. If this is not the case the query has to be broadcast to permit IDF aggregation. Also, the query is broadcast if none of the routing indexes contain applicable entries.

If an entry for query exists in the query-index, it is sent to the peers in this entry only, since no other have contributed to the top-k result for the current query. Otherwise, if the query category is in the category index, the query is sent to all peers to the corresponding category entry. In the case the even the categories-index does not contain a matching entry, the query needs to be broadcast. If the aggregation of the peer results is smaller than k, a broadcast of the query is done, too.

4.3.5.0.6 Index Update For each delivered result set, a query index entry is created, containing all peers and super peers which contributed to the result.

For the category index, one needs to know all peers holding documents of the specified category, even if they didn't contribute to the result set. Therefore, one collects this information as part of the result set, too, and use it to create category index entries.

As with the IDF index, the network volatility causes our routing indexes to become incorrect over time. The index entry expiration approach is used here as well to delete outdated entries.

CHAPTER V

Evaluation

In the previous chapter ProToRaDo is described. Proofs for Correctness and Optimality Results are given. This chapter presents the evaluation of the algorithm. The first part describes the idea behind simulations, the next section discusses existing systems. Afterwards the implementation of a simulation framework is described and simulation results are presented.

5.1 Simulator Design

5.1.1 Existing Simulators

This section gives an overview of other peer to peer simulation frameworks and motivates the implementation of the simulation framework presented in [108] and used here for evaluating ProToRaDo.

This section gives an overview of other peer to peer simulation frameworks and compares them to our work.

The **SimP2 simulator** [66] is designed to provide support and additional depth to an analysis of ad-hoc P2P-networks. The analysis is based on a non-uniform random graph model similar to Gnutella, and is limited to studying basic properties such as reachability and nodal degree. They leave out complex queries. On the other hand, SimP2 is very good for more detailed performance characteristics such as queuing delays and message loss.

3LS [122] is a discrete simulator using a central step-clock. It provides three levels: Network model, protocol model and user model. The network model uses a two dimensional matrix to define distance values between the nodes. The protocol model represents the P2P-protocol which should be investigated. Input can be simulated using the user model (which could be a interesting addition to our simulation-framework). Since 3LS is not efficient regarding memory usage, it is limited to rather small networks.

The authors of the **Packet-level Peer-to-Peer (PLP2P) Simulator** [63] state that one of the most important things in a simulation is the correct and mostly complete underlying network-structure. They assert that failure to consider low-level details can lead the simulation to inaccuracies. PLP2P provides a framework that can be used together with other simulators to achieve more accuracy in the simulations.

Narses Simulator [19] is a flow-based network simulator and thus does not concentrate on the packet-level to avoid the overhead of packet level simulators. To do this Narses offers a range of models that trade between fast runtimes and accuracy. Narses is therefore somewhere between packet level simulators and analytical models. Nevertheless the assumptions made by Narses are targeted towards reducing the complexity if simulations by approximations of physical aspects.

Evaluation Regarding our plan to simulate a new algorithm for search based on content and classification in a HyperCuP-topology, none of the current available simulators is capable of that, since they all concentrate on the traffic or information-flow on a much deeper level of the OSI-model. The observations are made directly from the transport-level or by making abstraction or assumptions on the (physical) aspects which are in contrast to our needs. Furthermore most simulators cannot be used to simulate different topologies with several parameters as needed it. To overcome these problems, a new simulator-framework which is described in the next section was implemented.

5.1.2 Simulator Implementation

The implementation is based on the discrete simulation framework SSF (Scalable Simulation Framework [40]). The Scalable Simulation Framework is an open standard of discrete event-simulations. The general layer is responsible for establishing the super peer topology and the connection between peers and super peers. Instead of using an IP network simulation as foundation, connections are specified by only two parameters, bandwidth (in number of messages per second) 2 and latency (in milliseconds). For both parameters average and deviation can be specified. The SSF provides an interface for discrete-event simulations supporting objectoriented models to utilize and extend the framework.

Extended the framework by this the potential for direct reuse of model code is maximized, while the dependencies on a particular simulator kernel implementation are minimized. The framework's primary design goal was to support high performance simulations and to make models efficient. The SSF provides several classes that are used to map the P2P-behavior to the model. The Entity is the central class in SSF. Entities can have processes for event-processing.

In the simulator the peers are implemented using entities. An Event changes the status of the system or is used for communication between entities. Regarding our simulator when use the events as messages between the peers. Processes are used to handle events during the simulation. An entity can have one or more processes. In- and out-channels are the communication channel between the entities. An entity can have several in- and out-channels, which are always connected 1 : 1. The configuration of the simulator is very simple using three XML-files which define the topology, duration of the simulation, time to live (TTL) for messages, number of peer, etc.

The TFxIDF calculation based on inverse indexes is done using the search engine Jakarta Lucene[1]. It is slightly modified, since it is needed to put IDF value in the calculation as

[1] http://www.jakarta.apache.org/lucene/docs/index.html

described in the algorithm discussion in the previous chapter.

5.1.2.1 Query- and Resource Description

The main functionality of the simulator is to experiment with query routing in super peer based peer to peer networks based on index information. Query messages consist of a list of keywords used to formulate the request as a simple conjunction together with a category in which should be searched. For the generation of such queries a configurable distribution is taken into account. The average number of keywords used in a query (and deviation)can be set, and the distribution used to chose terms (e.g. Zipf or uniform). When a peer is created, documents from an existing collection are assigned to it. When a query is received by a peer, an appropriate response set is generated.

5.1.2.2 Super Peer based Topology

As stated above the simulation framework assumes a super peer topology. All simple peers have exactly one connection to a super peer. The super peers form their own peer to peer network (it would also be possible to simulate a conventional peer to peer network by instantiating the super peer backbone only). The super peer network topology and protocol is pluggable. In contrast to other simulations the approach doesn't rely on a TCP/IP network simulation, but models connections between peers on a higher level. Connections are assigned a certain bandwidth (specified by messages per second) and delay (in msec). Both properties are modeled as normal distributions with configurable deviation. As it is assumed that SP/SP connections typically have a higher capacity than SP/P connections, the parameters can be set separately for these connection categories. Super peers are assumed to be highly available, so their up- and downtime is not modeled, but simulated using a static backbone. This makes it very simple to create different super peer topologies because it is not necessary to implement a full connection/disconnection protocol. Instead, a topology class creates all super peers and the connections between them on simulation startup.

5.1.2.3 Connections (Network Characteristics)

All connections are considered bi-directional. Each peer (including super peers) has an incoming message queue per connection, a single processing queue and an outgoing message queue per connection. Messages between the peers are interpreted as discrete events.

5.2 Evaluation: Content

The query processing for top-1 and top-10 queries was evaluated, both in a static and in a dynamic network setting.

5.2.1 Data Setup

For the experiments the network size varied between 100 and 2000 peers (connected via 2 to 16 super peers), each holding 50 documents on average (with standard deviation 10).

The LA Times articles from the TREC-CD 5 were used, the documents were distributed randomly among the peers. This is obviously the worst case for the approach: when documents are clustered on peers, the number of addressed peers can be essentially reduced.

Keyword queries with query term count average 2.0 and standard deviation 1.0 were posed. For these queries, terms from the documents were selected randomly. A a Zipf-shaped query frequency distribution with skew-factor 1.0 is assumed (see figure 5.1). The queries were pre-generated for occurrence frequencies >1% (head)). Below this threshold queries were generated randomly with uniform distribution. The originating peer was selected randomly (tail).

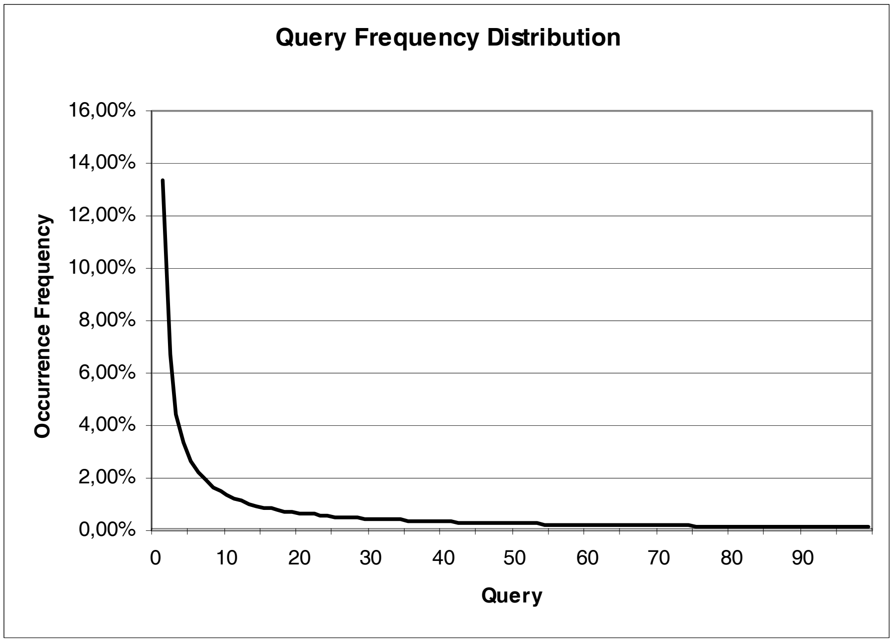


Figure 5.1: Zipfian Query Distribution

5.2.1.1 Top-10

The top-10 case is the typical search engine use case. In most cases, users expect to find an appropriate document among the best 10 hits. Therefore it is a reasonable value for the number of requested hits. If no suitable documents are found, users either refine their query (covered by the scenario) or ask for more results (not included in the simulation, but of course possible within our approach).

5.2.1.2 Top-1

This situation often occurs, when applications or services execute a query to use the results in an internal process. A typical example can be found in the context of Web Services. To execute a predefined process consisting of several Web Services, the process execution engine has to find best matches for each service specification used in the process specification. However, only the best match is relevant, because in automated process execution only this service will be called as part of a workflow. These searches apply to both, necessary service discovery operations, e.g. [5], and the actual selection of a semantically best matching service like in [6].

5.2.1.3 Static scenario

To evaluate how efficient the index works and how fast it builds up, the first scenario was a static one where all peers are set up before the simulation starts. This scenario has practical applications e.g. for a network of service registries where registry nodes join and leave very rarely.

5.2.1.4 Dynamic scenario

Obviously, it was also interesting how good the algorithm works in a volatile network, the typical peer to peer case. To simulate volatility, each peer was given a lifetime (with normal distribution). After its life-time, the peer leaves the network, and soon afterwards a new peer with new documents joins. The lifetime was adjusted so that during a simulation run about 20% of the peers left the network and were replaced by new peers. In this scenario the indexes need to updated. As described in section 3, each index entry gets an expiration time on creation, based on a specified expiration period. This ‘best before’ date will guarantee that our indexes do not age too much and provide out-dated information.

5.2.2 Hypotheses

1. After the routing index is built up, not more than $n_{SP}+k$ peer nodes will be touched, where n_{SP} is the number of super peers.
2. On average, the difference between the top-k document set delivered with ProToRaDo and the document set which would be retrieved by a central computation of the union of all peers documents is small (i.e. the index based routing also in volatile peer to peer networks only causes slight changes).

5.2.3 Results

5.2.3.1 Static scenario

Figure 5.2 shows how the index develops as more and more queries are processed by the network. The simulation starts with an empty index. As described, each query is added to the index on closing of the transaction. Thus the number of already indexed queries increases continuously. If a new query is sent, the algorithm checks if it is already in the index. The following figures show how many of the queries were already in the index (as floating average with an interval of 200). The index coverage increases quickly. After 550 queries more than 90% of all queries are found in the index. After about 550 queries the index ratio stays rather constant, with some random variations, as shown in figure 5.3.

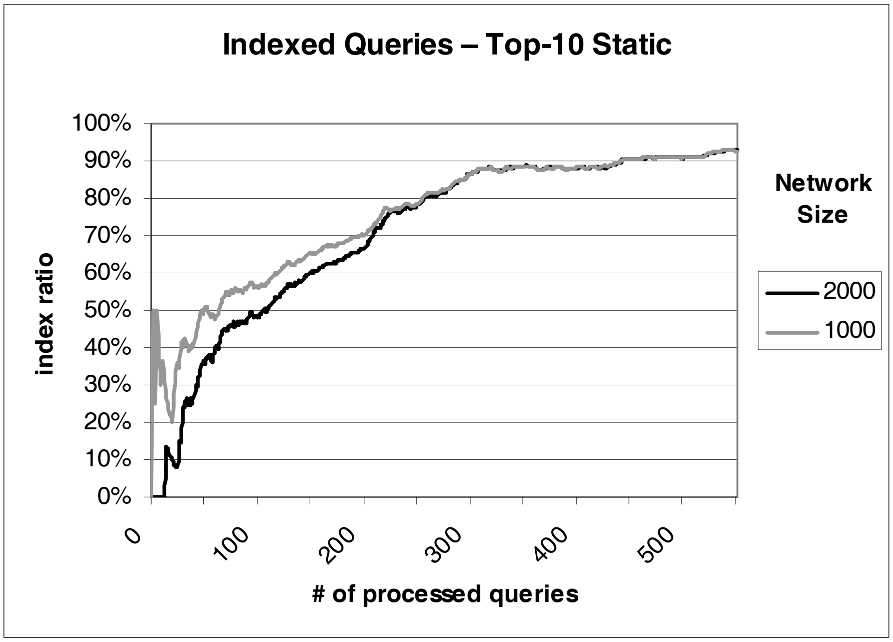


Figure 5.2: Index Development (first 550 queries)

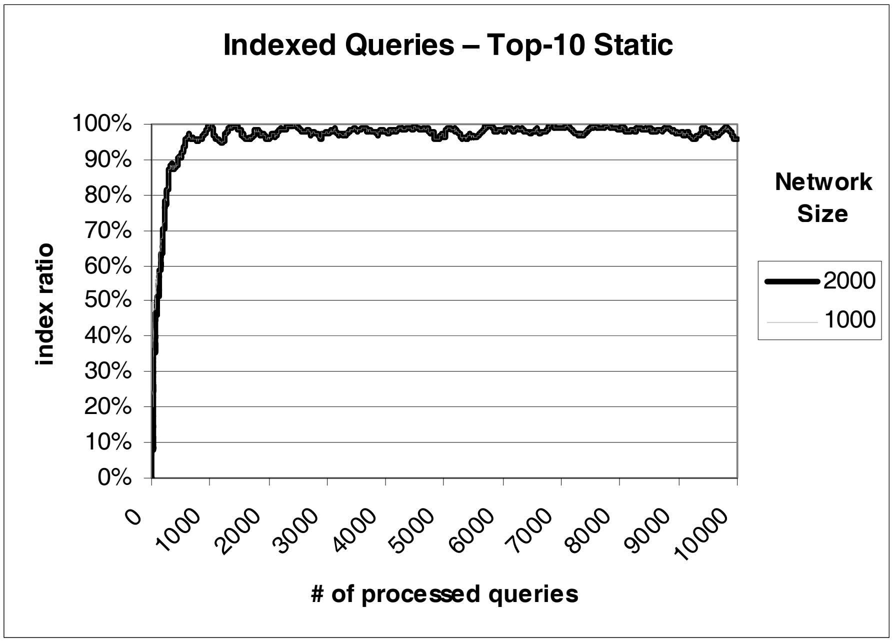


Figure 5.3: Index Development (10000 queries)

The rapid build up of the index corresponds to a significant reduction of contacted peers (see figure 5.4). After 1000 queries the average number of contacted peers has become relatively constant (from 10 for 100 peers to 49 for 2000 peers). Therefore, only the first 2000 queries are shown. The top-1 case shows the same trend. As can be expected, the average of contacted peers is a little lower (after 2000 queries it ranges from 5 for 100 peers to 41 for 2000 peers). However, the simulation results show a smaller difference than could be expected. The number of messages sent to evaluate a query is proportional to the number of contacted peers. This is illustrated in figure 5.5 for the top-1 static case.

5.2.3.2 Dynamic scenario

Figure 5.6 shows the average number of contacted peers for the top-1 dynamic case. While it is slightly higher than the top-1 static results, the general index performance is not significantly affected by the introduction of an expiration period for the index entries forcing periodic broadcasts for all indexed terms to get a new snapshot of the changed peer to peer network in the index. On average the number of contacted peers in the dynamic case will increase by a few percent, e.g. in the top-1 case the difference between static and dynamic is only 6%.

The curves exhibit a slight waveform. Besides random fluctuations this is caused by the way the expiration works. Frequently posed queries are indexed shortly after simulation start. They expire all in the same time frame, causing the first broadcast wave. Over time,

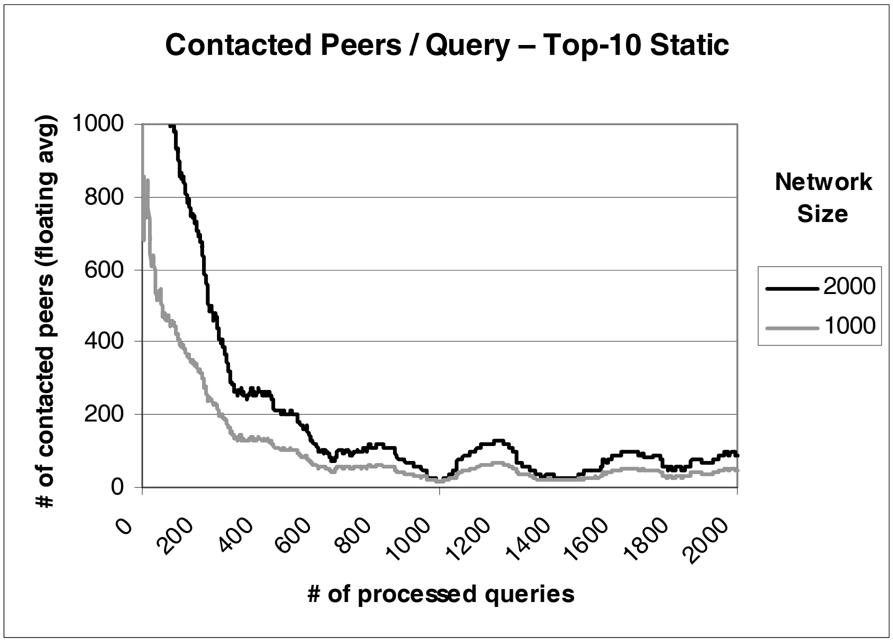


Figure 5.4: Contacted Peers per Query

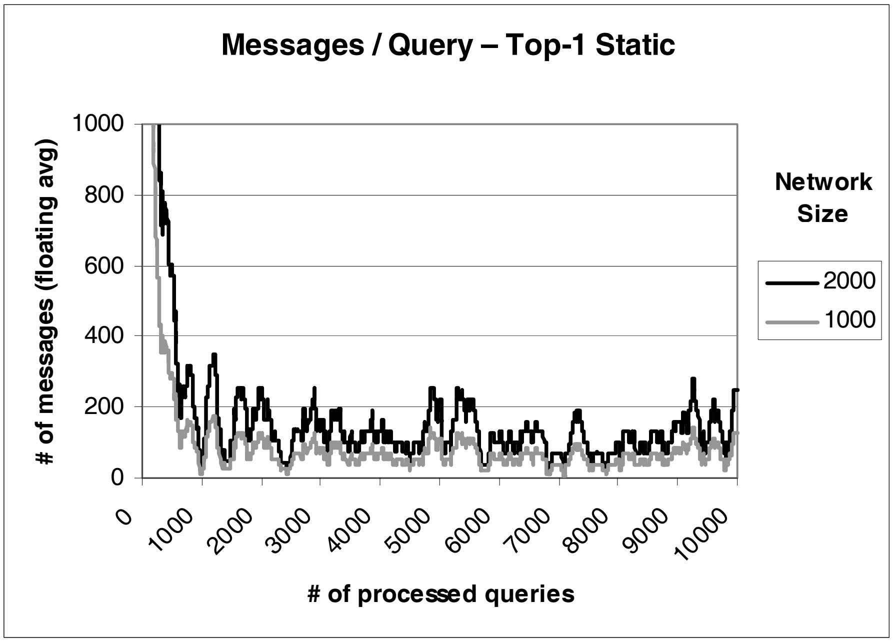


Figure 5.5: Message per Query (top-1, static)

these waves get less significant due to the probabilistic query frequency distribution. In reality, such waves wouldn't occur because node number and query frequency would grow over time instead of starting off with a fixed amount, thus also causing a less steep increase

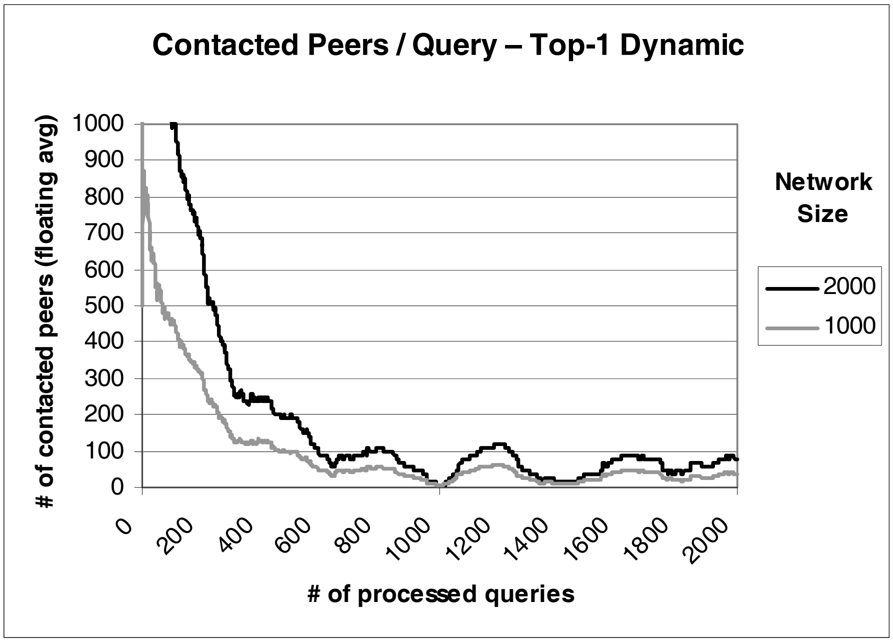


Figure 5.6: Contacted Peers per Query (top-1, dynamic)

of index content.

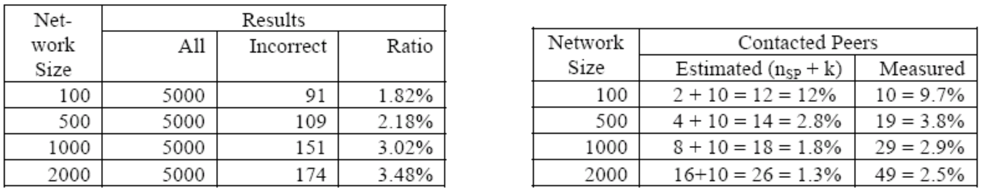
5.2.4 Discussion

Coming back to the two initial hypotheses the experiments allow to state the following confirming results:

Figure 5.7(a) shows for how many queries the computed result differs from the 'perfect' result (which is obtained by evaluating the same query against the joint document collection of all currently existing peers). Though network changes can temporarily invalidate index entries, this is corrected as soon as affected entries expire. The experiments show the ratio of incorrect results to remain sufficiently low, rising only slightly with network sizes. Please note that *all* differences between the simulated and the perfect result were count as being 'incorrect' retrievals. However, such incorrect results may still be quite good matches, just not the best possible as given by centralized approaches using expensive broadcasts.

5.2.4.1 Estimated vs. measured contacted peers

- Figure 5.7(b) presents the expected and actual ratio of contacted peers for the top-10 case. The last column contains the average contacted peers of queries 2001 to 10000. The first 2000 queries are regarded as initial phase until the simulation stabilizes and



Net-work Size	Results		
	All	Incorrect	Ratio
100	5000	91	1.82%
500	5000	109	2.18%
1000	5000	151	3.02%
2000	5000	174	3.48%

(a) Ratio of incorrect hits

Network Size	Contacted Peers	
	Estimated (n_{SP} + k)	Measured
100	2 + 10 = 12 = 12%	10 = 9.7%
500	4 + 10 = 14 = 2.8%	19 = 3.8%
1000	8 + 10 = 18 = 1.8%	29 = 2.9%
2000	16+10 = 26 = 1.3%	49 = 2.5%

(b) Estimated vs. measured contacted peers

Figure 5.7: Results

are thus left out. It shows that the estimated numbers of contacted peers is not fully reached. The reason is because, though a large percentage of queries gets indexed very quickly, there constantly remains a small amount of queries that still have to be broadcasted. The impact of this phenomenon depends on the network size. It is caused by our query generation strategy which (following the assumption of a Zipf distribution) creates random queries for occurrence frequencies below 1%. Still the ratio of contacted peers/network size is very low, and the number of connected peers increases sub-linearly with increasing network size.

- The assumption that the approach works nearly as well in a volatile network as it does in a static network can also be confirmed, as shown by the results in our dynamic scenarios. Though in the practical tests it was assumed quite a high volatility (20% of the network dropping out and being replaced), in all cases the number of contacted peers increased only by a few percent over the static case, and remained relatively stable during the simulation. The indexing scheme is thus applicable for most practical environments.

5.3 Evaluation: Content and Classification

5.3.1 Data Setup

The TREC document collection volume 5 consisting of LA Times articles was used for the experiments. The articles are already categorized according to the section they appeared in, and the information is used as base for the document classification. To simulate a network of document providers, these articles are distributed among the peers in the network. The simulated network consists of 2000 peers, each providing articles from three categories on average (with a standard deviation of 2.0). A Zipf-distribution for query frequencies with skew of -0.0 is assumed. News articles are popular only for a short time period, and the request frequency changes correspondingly. With respect to the Zipf-distribution this means that the query rank decreases over time. Query terms were selected randomly from the underlying documents. In the simulation, 200 new most popular queries every 2000 queries are generated which supersede the current ones and adjust query frequencies accordingly. This shift may be unrealistically high, but serves well to analyze how our algorithm reacts to such popularity changes.

5.3.2 Results

5.3.2.1 Index size

Figure 5.8 shows how the IDF index at each super peer grows over time. After 10000 queries it has grown to a size of 2015, only a small fraction of all terms occurring in the document collection. A global inverted index would have had contained 148867 terms). This clearly underlines that much effort can be saved when only indexing terms which are actually appearing in queries.

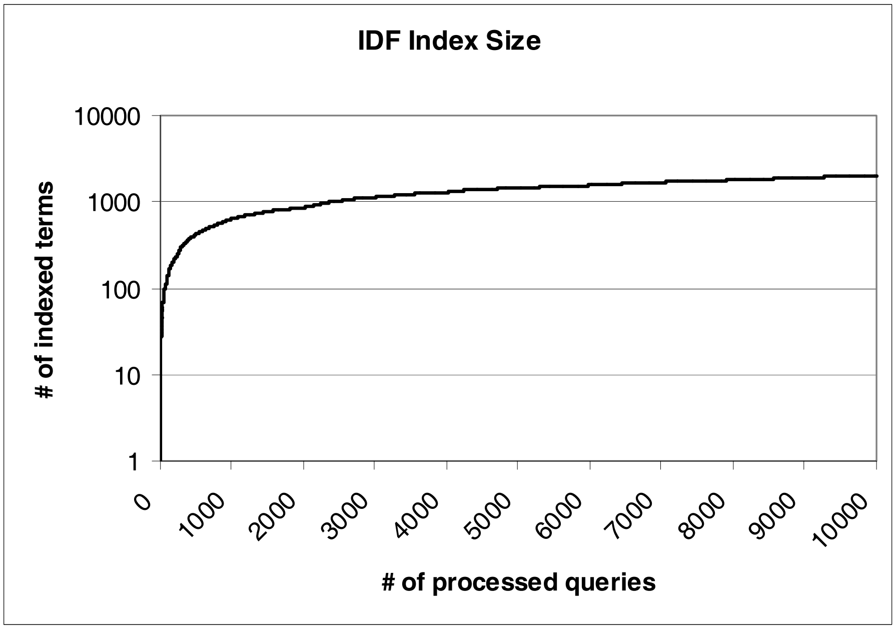


Figure 5.8: Index size

5.3.2.2 Index Effectivity

Both category and query index become quite effective. After nearly 2000 queries, the query index achieves a coverage of 80%. Figure 5.9 shows how each popularity shift causes a coverage reduction from which the query index recovers after about 1000 queries. This shows that a change in query popularity over time is coped with after a very short while.

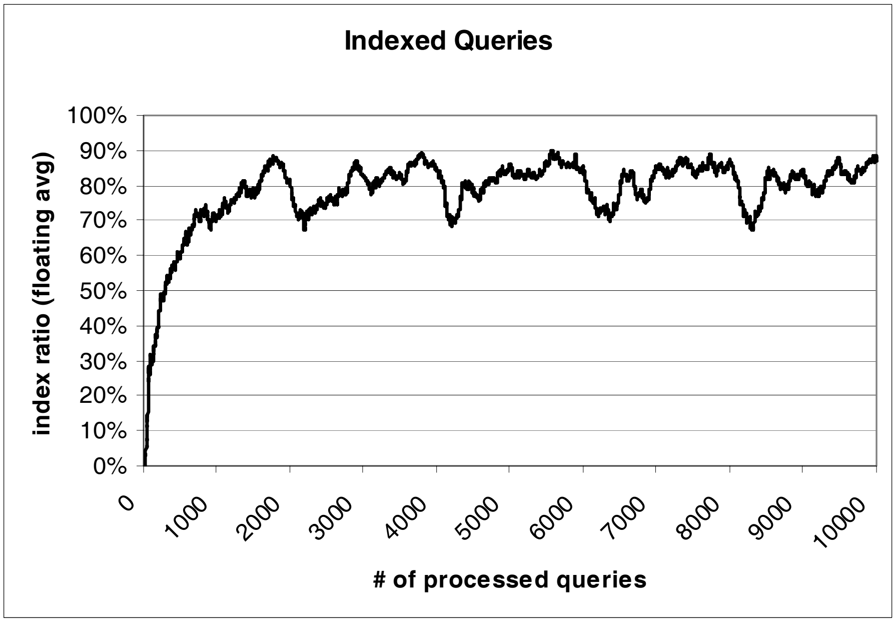


Figure 5.9: Coverage of query index

As there are only about 120 different categories, after less then 1000 queries the index contains nearly all of them (figure 5.10). It is assumed that news provider specialized on some topics change these topics only very infrequently. Therefore, peers do not shift their topics during the simulation. Thus, the category index serves to reduce the number of contacted peers continuously, also after popularity shifts.

Figure 5.11 shows how many peers had to be contacted to compute the result. The influence of popularity shifts on the whole outcome can also be seen clearly. The category index takes care that the peaks caused by popularity shifts don't become too high. Summarized, the combination of both indexes yields a high decrease of contacted peers compared to broadcasting.

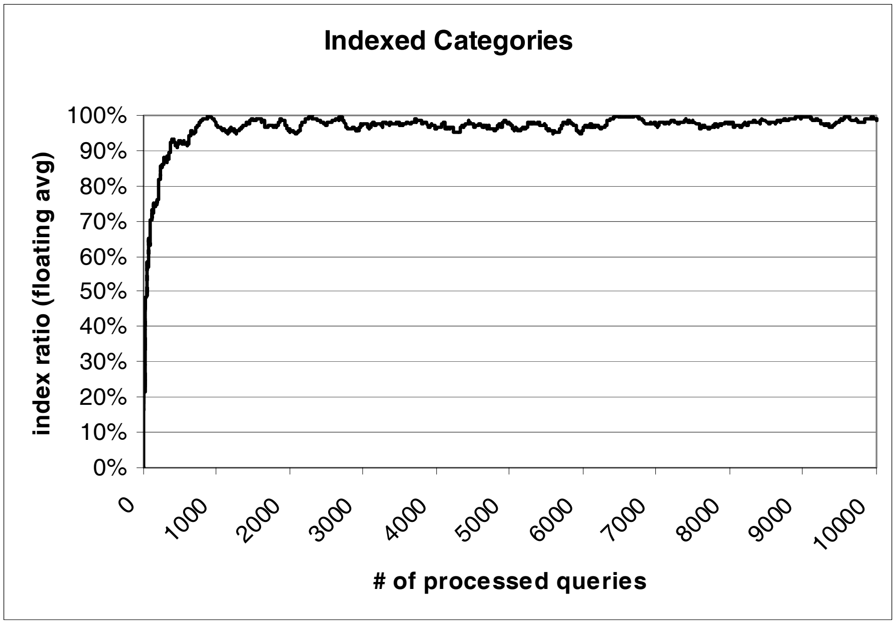


Figure 5.10: Coverage of category index

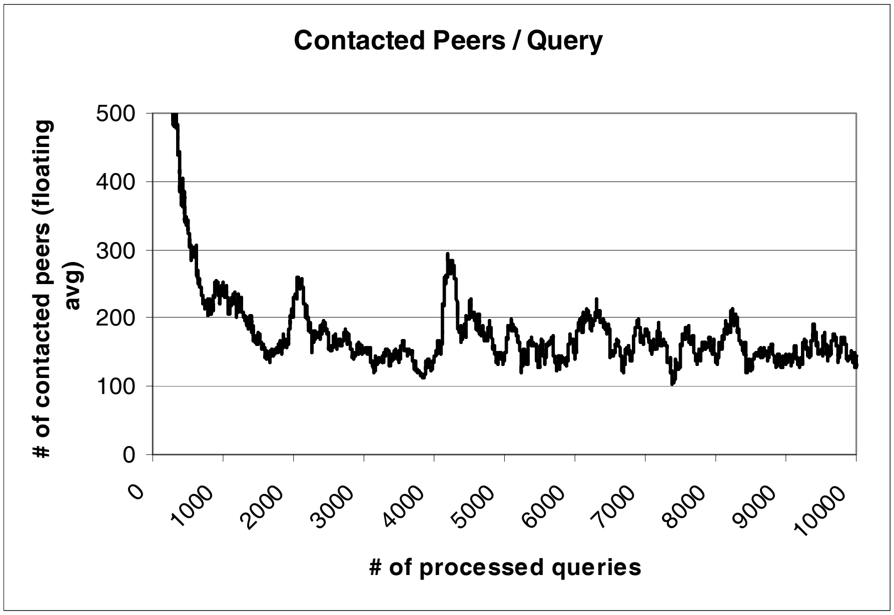


Figure 5.11: Contacted peers per query

CHAPTER VI

Summary and Outlook

6.1 Summary

This thesis introduces a new algorithm for ranked retrieval in peer to peer networks. Ideas and methods from the domains of databases, information retrieval and peer to peer are integrated:

The infrastructure is peer to peer, where peers are connected as nodes which have all nearly the same rights and liabilities in the network, independent of any centralized instance storing information about the whole network. In the information retrieval area several very good methods for ranked retrieval were presented over the years. They are discussed in this thesis and the problems arising in the distributed environment are discussed, i.e. collection wide information. From the database community the idea of top-k is adapted to peer to peer environments. After ranking the results for a posed query, it is an important step to reduce the maximal number of entries in the result set, since user are only interested in a limited number of best results. The new algorithm ProToRaDo introduced in this thesis offers distributed ranking with respect to collection wide information.

Existing retrieval approaches are presented and their drawbacks are discussed (e.g. gossiping to maintain an inverted index for a document collection or ignoring collection wide information).

ProToRaDo is query-driven. This means that computations for the result set creation is only done for a specific query. No index is computed in advance for all documents, which would lead to high traffic for maintaining it over time. If no query is posed, ProToRaDo adds to traffic to the network, even

On the peers, any method for result ranking can be used. Of course, here the tradeoff between results accuracy and additional load on the peers doing the computation must be taken into account.

Correctness and optimality results of ProToRaDo are shown. This means it is ensured that the algorithm always returns the best k resources for a particular query and that the algorithm needs to transfer only a minimum amount of object data to find the correct result set.

A simulation framework was implemented and used for concrete data setups. Several parametrization scenarios were used to verify the algorithm. Because of its properties, a peer to peer network cannot have the same optimality as a centralized system with an instance knowing important information about the whole network. Thus, both from networking view and from information retrieval view, ProToRaDo needs to be measured how good it is compared to the systems with optimal information. The results clearly show the improvements regarding the message count needed to receive the top-k resources, the reduction of traffic in the peer to peer network and the feasibility of ProToRaDo even in volatile networks, where the algorithm can be used as well by adapting the parameters for the index updates. From a network point of view, a global index in a volatile network is not usable for two reasons:

1. Traffic: A global index containing all words of the document collection grows very fast. This makes it nearly impossible to distribute it over a peer to peer network. Even reducing the amount of data transferred at once using gossiping does not solve the problem.
2. Volatility: If the network is not static, it gets even worst since the global index must be distributed very fast in the network to ensure its correctness.

Both aspects are handled optimal in ProToRaDo, where no global index is stored and thus no continuous index updates are needed. Only for terms which are queried, the algorithm produces very little traffic to route the query and to update the collection wide information stored at the super peers; the indexes at the super peers do only maintain entries for terms that were used in a particular query which ensures a nearly optimal index maintenance without redundant or unused data.

6.2 Outlook and Future Work

The work presented can be seen both as a starting point for further improvements and as base for new developments. This section describes the three most interesting aspects of possible work in the future; the named aspects are all current research work in progress.

Result merging In our current Top-k-approach we assume that all peers use the same scoring function and do not fake the results they deliver for a posed query.

In the next step we face the following challenge: Peers might be interested in faking results to become more important in general or for specific topic(s). Furthermore we do not longer assume that all peers use the same scoring function. Thus, the second question is if a peer is malicious or uses a different scoring function.

Basic idea of calibration: If a peer joins a network a calibration process will check if scores of documents differ in a "not common" way. If so we want to tag the delivering peer with a factor/weight for future result combination. Open questions are: Do we need to find a threshold to define if a peer is far away from a common score? Can we use the majority of the results to be taken as normal?

We assume three different kinds of being malicious:

1. The peer returns a faked score.
2. The peer returns a document, which has be altered to be best fitting the query, e.g. adding important words to the text.
3. The peers fakes the document frequencies. This will not affect the current answer, but should be usable to become more important for one topic over time.

First ideas to approach the challenges exist for 1 and 2:

ad 1) Do the scoring again at the SP

ad 2) Similarity of documents

The next steps are: Test if there are similarities between pairwise comparisons of documents to detect values which do not fit.

Community Collaboration of people (e.g. researchers) is a promising direction for peer to peer. Currently, we are working on the following topic: We assume that more and more meta data will be available for particular areas. Focusing on the collaboration in a scientific peer to peer network one can think of meta data describing papers that each user has on his or her desktop. Those annotations can range from information like at which conference it was published to very personal opinions about a paper in a specific context.

Researchers can benefit from such annotations when they find a community (or interest group), in which people work on similar topics and thus can probably provide additional information. The main question is how a user can find such a community, to which he can contribute and from which he can benefit by sharing information that are in the same context.

Our idea is to split this into two steps. In the first step we want to use our top-k algorithm to find people in the network who have documents which might be interesting to a particular topic. Our plan is to do this by finding those documents which have an overlap with the document collection of the searching user. The overlap can be thought of as the number of important words which appear in both the user's collection and the collections of other user in the whole peer to peer network. We want to use methods from information retrieval to investigate which terms in the user's document collection are representing his interests.

We can use the top-k algorithm: Having those words we can use them as query terms. ProToRaDo can the return the resources which match best against the interests of the user. Assuming that the hosts of the resources are not anonymous, they are the people which build an interest group for a specific context.

The second step is not directly related with ProToRaDo since it concentrates on how to minimize the amount of data used for representation of the meta data, i.e. using RDF (Resource Description Framework). This would allow inference with predictable time consumption. But nevertheless, even in that part of the work it might be needed to store information in a distributed index.

Index usability The indexes used in ProToRaDo can be seen as caches for the queries and corresponding data for the terms. Each entry in the index has a best before date that indicates if the entry must be updated to is taken as still valid for a particular query. In a static network where peers (documents) do not change, this is no challenge. But in volatile networks the TTL of an index entry is strongly connected with the quality of the results, since old entries route the query not to new nodes which potentially have even better results. The idea how we approach this is as follows: To create an index entry, we need to broadcast once, which costs $n + m - 1$ messages (n = number of peers, m = number of super peers). To process a query without index entry, we also need to broadcast it (cost: $n + m - 1$ messages). To process a query with index entry, we have to forward it to at most k peers. In the worst case, the query has to be sent to all super peers, so we need at most $k + m - 1$ messages. So the minimal gain is $(n + m - 1) - (k + m - 1) = n - k$. When we update each index after the xth occurrence of the respective query, the index entry gain is $n - k - \frac{n+m-1}{x}$ per query (=index entry usage).

An index entry becomes useless if the gain becomes less than 0. We gain something iff $n - k - \frac{n+m-1}{x} > 0 \equiv x > \frac{n+m-1}{n-k}$. With a network change rate of c per query, we assume that after one query an error rate of c is introduced into every result by outdated index entries (incorrect TopPeer list). We model the error rate f_q of a peer index as exponential decay. Directly after an index update, the error rate of correct peers is $1 - (1 - c)^0 = 0.0$. After y queries, it is $1 - (1 - c)^y$. Having a zipfian query distribution and taking the expectation for queries into account, we get that the error rate is $1 - (1 - c)^{e_q \cdot v}$.

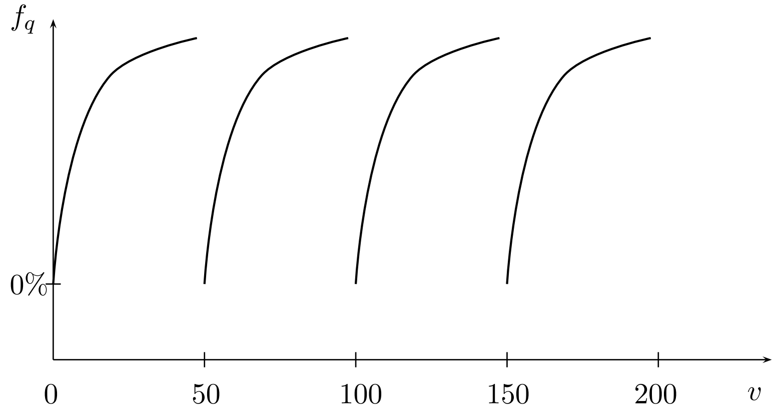


Figure 6.1: Development of error rate over time

First results show that our (yet simple) mathematical model leads to the results that one expects intuitively for f_q when biasing x and c: if e_q gets smaller the error grows, because it takes a longer time until the index gets refreshed. Currently, we investigate how x and c correspond to each other, and which values are valid for realistic assumptions. This will probably lead to a more complex model that can be used to model a cost benefit analysis.

APPENDICES

APPENDIX A

Curriculum Vitæ

Uwe Thaden, geboren am 16. Juni 1973 in Börger

1979 – 1983	Grundschule Voslapp Wilhelmshaven
1983 – 1985	Orientierungsstufe Nogatstraße, Wilhelmshaven
1985 – 1989	Agnes Miegel-Realschule, Wilhelmshaven
1989 – 1992	Technisches Gymnasium, Wilhelmshaven
Mai 1992	Abitur
1992 – 1993	Grundwehrdienst
1993 – 1998	Studium der Informatik an der Carl von Ossietzky Universität Oldenburg
Mai 1998	Diplom der Informatik (Dipl.-Inform.)
1998 – 2000	Angestellt bei der TUI InfoTec, Hannover
Seit Oktober 2000	Wissenschaftlicher Mitarbeiter. Institut für Technische Informatik, Abteilung Rechnergestützte Wissensverarbeitung und am Forschungszentrum L3S, Universität Hannover
10. August 2005	Einreichung dieser Dissertation (Fakultät für Elektrotechnik und Informatik der Universität Hannover)
12. Dezember 2005	Promotion zum Dr. rer. nat. an der Fakultät für Elektrotechnik und Informatik der Universität Hannover

APPENDIX B

Publications

1. *DL meets P2P - Distributed Document Retrieval based on Classification and Content*
   Co-authors: Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski
   ECDL, 2005 [21]

2. *Caching for Improved Retrieval in Peer-to-Peer Networks*
   Co-authors: Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski
   GI/ITG-Workshop, 2005 [20]

3. *Here is the News - Distributed Document Retrieval based on Classification and Content*
   Co-authors: Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski
   Technical Report, 2005

4. *Progressive Distributed Top k Retrieval in Peer-to-Peer Networks*
   Co-authors: Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski
   ICDE, 2005 [22]

5. *Top-k Query Evaluation for Schema-Based Peer-to-Peer Networks*
   Co-authors: Wolfgang Nejdl, Wolf Siberski, Wolf-Tilo Balke
   ISWC, 2004 [90]

6. *A Simulation framework for schema-based query routing in P2P-networks*
   Co-authors: Wolf Siberski
   P2P&DB, 2004 [108]

7. *iMOL: Ein experimentelles Werkzeug zur interaktiven Validierung von Softwaremodellen*
   Co-authors: Tobias Buchloh, Friedrich Steimann
   Modellierung, 2004 [119]

8. *Semantic Overlay Clusters within Super-Peer Networks*
Co-authors: Alexander Lser, Felix Naumann, Wolf Siberski, Wolfgang Nejdl
DBISP2P (VLDB colocated workshop), 2003 [81]

9. *Intelligently Authoring Metadata for a*
Semantic Web Peer-to-Peer Environment
Co-authors: Jan Brase, Wolfgang Nejdl, Mark Painter, Michael Sintek
Technical Report, 2003 [29]

10. *Animated UML as a 3d-illustration for teaching OOP*
Co-authors: Friedrich Steimann
ECOOP, 2003 [121]

11. *A Semantic Web based Peer-to-Peer Service Registry Network*
Co-authors: Wolf Siberski, Wolfgang Nejdl
Technical Report, 2003 [120]

12. *Proposing Mobile Pair Programming*
Co-authors: Friedrich Steimann, Jens Gößner,
OOPSLA Workshop on Pair Programming, 2002 [111]

13. *Animiertes UML als Medium für die Didaktik*
der objektorientierten Programmierung
Co-authors: Friedrich Steimann, Wolf Siberski, Wolfgang Nejdl,
Modellierung, 2002 [112]

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Dublin core metadata initiative. http://www.dublincore.org/.

[2] edusplash. http://www.edusplash.net/.

[3] Google help: Basics of search. http://www.google.com/intl/en/help/basics.html.

[4] IEEE LTSC, WG12: Learning object metadata (LOM). http://ltsc.ieee.org/wg12/.

[5] Porter stemmer. http://www.tartarus.org/ martin/PorterStemmer/.

[6] RFC 3056 - connection of IPv6 domains via IPv4 clouds. http://www.faqs.org/rfcs/rfc3056.html.

[7] Zipf, power-laws, and pareto - a ranking tutorial by Lada A. Adamic. http://www.hpl.hp.com/research/idl/papers/ranking/ranking.html.

[8] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *Proceedings of the 9th International Conference on Cooperative Information Systems (CoopIS)*, pages 179–194, 2001.

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

[10] K. Aberer, P. Cudré-Mauroux, and M. Hauswirth. The chatty web: Emergent semantics through gossiping. In *Proceedings of the 12th International World Wide Web Conference (WWW)*, 2003.

[11] K. Aberer and J. Wu. A framework for decentralized ranking in web information retrieval. In *Proceedings of the fifth Asia Pacific Web Conference (APWeb2003)*, 2003.

[12] R. L. Ackoff. From data to wisdom. *Journal of Applied Systems Analysis*, 16(1):3–9, 1989.

[13] L. A. Adamic and B. A. Huberman. Zipf's law and the internet. *Glottometrics*, 3, 2002.

[14] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman. Search in Power-law Networks. In *Physical Review E, 64 46135*, 2001.

[15] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *Proceedings of the Second Conference on Innovative Data Systems Research*, 2003.

[16] S. Androutsellis and T. Diomidis-Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371, 2004.

[17] N. B. Azzouna and F. Guillemin. Analysis of adsl traffic on an ip backbone link. *Globcom*, 7:3742–3746, 2003.

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

[19] M. Baker and T. Giuli. Narses: A scalable flow-based network simulator. Stanford University Technical Report, 2002.

[20] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Caching for improved retrieval in peer-to-peer networks. In *Proceedings of the GI/ITG-Workshop "Peer-to-Peer-Systeme und -Anwendungen"*, 2005.

[21] 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 9th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2005)*, 2005.

[22] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks. In *Proceeding of the 21st International Conference on Data Engineering (ICDE)*, 2005.

[23] D. Barkai. Technologies for sharing and collaborating on the net. In *Proceeding of the 1st International Conference on Peer-to-Peer Computing (P2P 2001)*, 2001.

[24] L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The google cluster architecture. In *IEEE Micro*, volume 23, pages 22–28. IEEE, 2003.

[25] M. Bawa, G. Manku, and P. Raghavan. Sets: Search enhanced by topic segmentation. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, 2003.

[26] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. MINERVA: Collaborative P2P search (demo). In *Proceedings of the 31st International Conference on Very Large Databases (VLDB)*, 2005.

[27] M. Bender, S. Michel, C. Zimmer, and G. Weikum. The MINERVA project: Database selection in the context of P2P search. In *Datenbanksysteme in Business, Technologie und Web (BTW2005)*, 2005.

[28] M. Berry, Z. Drmac, and E. Jessup. Matrices, vector spaces, and information retrieval. *SIAM review*, 41(2):335–362, 1999.

[29] J. Brase, W. Nejdl, M. Painter, M. Sintek, and U. Thaden. Intelligently authoring metadata for a semantic web peer-to-peer environment. Technical report, Learninglab Lower Saxony, 2003.

[30] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30:107–117, 1998.

[31] R. R. Brooks, M. Zhu, J. Lamb, and S. S. Iyengar. Aspect-oriented design of sensor networks. *Journal on Parallel Distributed Computing*, 64(7):853–865, 2004.

[32] N. Bruno, L. Gravano, and A. Marian. Evaluating top-k queries over web-accessible databases. In *Proceedings of the 18th International Conference on Data Engineering*, San Jose, CA, USA, 2002. IEEE Computer Society.

[33] V. Bush. As we may think. 1945. Originally published in the July 1945 issue of The Atlantic Monthly; available at: http://www.ps.uni-sb.de/∼duchier/pub/vbush/vbush-all.shtml.

[34] L. F. Cabrera, M. B. Jones, and M. Theimer. Herald: Achieving a global event notification service. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001.

[35] M. Cai and M. Frank. RDFPeers: a scalable distributed rdf repository based on a structured peer-to-peer network. In *Proceedings of the 13th international conference on World Wide Web (WWW)*, pages 650–657, 2004.

[36] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. Technical Report MSR-TR-2002-82, Microsoft Research, One Microsoft Way, Redmond, WA 98052, 2002.

[37] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 407–418, 2003.

[38] Q. Chen. The origin of power laws in internet topologies revisited. In *21st Annual Joint Conference of the IEEE Computer and Communications Societies*, 2002.

[39] E. F. Codd. A relational model of data for large shared data banks. In *Communications of the ACM*, volume 13, pages 377–387. ACM, 1970.

[40] J. Cowie, H. Liu, J. Liu, D. Nicol, and A. Ogielski. Towards realistic million-node internet simulations. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, Jun 1999.

[41] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings International Conference on Distributed Computing Systems*, July 2002.

[42] A. Crespo and H. G. Molina. Semantic overlay networks for P2P systems. Technical report, Stanford University, 2003.

[43] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In *Twelfth IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*. IEEE Press, 2003.

[44] V. K. D. Zeinalipour-Yazti and D. Gunopulos. Information retrieval techniques for peer-to-peer networks. *IEEE CiSE Magazine, Special Issue on Web Engineering*, 30(4):12–20, 2004.

[45] V. K. D. Zeinalipour-Yazti and D. Gunopulos. Exploiting locality for scalable information retrieval in peer-to-peer systems. *Information Systems Journal*, 30(4):277–298, 2005.

[46] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Widearea cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, 2001.

[47] H. Darwen and C. Date. The third manifesto. In *ACM SIGMOD*, volume 24, pages 39–49. ACM, 1995.

[48] N. Daswani, H. Garcia-Molina, and B. Yang. Open problems in data-sharing peer-to-peer systems. In *Proceedings of 9th International Conference on Database Theory (ICDT 2003)*, 2003.

[49] J. L. Dawson. Suffix removal for word conflation. *Bulletin of the Association for Literary & Linguistic Computing*, 2:33–46, 1974.

[50] P. Druschel and A. Rowstron. Past: Persistent and anonymous storage in a peer-to-peer networking environment. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS)*, pages 65–70, 2001.

[51] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM Computer Communication Review, Vol 29(4)*, 1999.

[52] I. Foster. What is the grid? a three point checklist. Technical report, 2002. http://www-fp.mcs.anl.gov/ foster/Articles/WhatIsTheGrid.pdf.

[53] I. T. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *Peer-to-Peer Systems II, Second International Workshop, IPTPS*, pages 118–128, 2003.

[54] P. Ganesan, P. K. Gummadi, and H. Garcia-Molina. Canon in g major: Designing dhts with hierarchical structure. In *Proceedings of the 24th International Conference*

on Distributed Computing Systems (ICDCS 2004), pages 263–272. IEEE Computer Society, 2004.

[55] P. Ganesan, Q. Sun, and H. Garcia-Molina. Adlib: A self-tuning index for dynamic p2p systems. Technical report, 2004.

[56] P. Ganesan, Q. Sun, and H. Garcia-Molina. Adlib: A self-tuning index for dynamic p2p systems. In *Proceedings of the 21 International Conference on Data Engineering (ICDE)*, pages 256–257, 2005.

[57] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP)*, pages 29–43. ACM, 2003.

[58] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2), 2002.

[59] G. Glider. Metcalfe's law and legacy. *Forbes ASAP*, 1993.

[60] L. Gong. Jxta: A network programming environment. *IEEE Internet Computing*, 2001.

[61] U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing multi-feature queries for image databases. In *Proceedings of 26th International Conference on Very Large Data Bases(VLDB)*, pages 419–428, 2000.

[62] P. Haase, J. Broekstra, M.Ehrig, M. Menken, P.Mika, M. Plechawski, P. Pyszlak, B. Schnizler, R. Siebes, S. Staab, and C. Tempich. Bibster - A Semantics-Based Bibliographic Peer-to-Peer System. In *Proceedings of the 3rd International Semantic Web Conference (ISWC)*, 2004.

[63] Q. he, M. Ammar, G. Riley, H. Raj, and R. Fujimoto. Mapping peer behavior to packet-level details: A framework for packet-level simulation of peer-to-peer systems. In *MASCOTS*, 2003.

[64] R. Huebsch, J. M. Hellerstein, N. L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier.

[65] K. S. Jones and P. Willett, editors. *Readings in Information Retrieval.* Morgan Kaufmann, 1997.

[66] K. Kant and R. Iyer. Modeling and simulation of ad-hoc/p2p file-sharing networks. In *Tools 2003*, 2003.

[67] F. B. Kashani and C. Shahabi. SWAM: a family of access methods for similarity-search in peer-to-peer data networks. In *Proceedings of the 2004 ACM International Conference on Information and Knowledge Management (CIKM)*, pages 304–313, 2004.

[68] P. Keleher, B. Bhattacharjee, and B. Silaghi. Are virtualized overlay networks too much of a good thing. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[69] A. Kent, Berry, Luehrs, and Perry. Operational criteria for designing information retrieval systems. *American Documentation*, 6(2):93–101, 1955.

[70] D. E. Knuth. *The Art of Computer Programming, Vol. III, Sorting and Searching.* Addison-Wesley, 1998.

[71] R. Krovetz. Viewing morphology as an inference process. In *Proceedings of the 16th ACM SIGIR Conference*, pages 191–202, 1993.

[72] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ASPLOS*, 2000.

[73] M. Lesk. The seven ages of information retrieval. In *Proceedings of the Conference for the 50th anniversary of As We May Think*, pages 12–14, 1995.

[74] B. Leuf. *Peer to Peer: Collaboration and Sharing on the Internet.* Addison-Wesley, 2002.

[75] Y. Li, Z. A. Bandar, and D. McLean. An approach for measuring semantic similarity between words using multiple information sources. *IEEE Transactions on Knwoledge and Data Engineering*, 15(4), 2003.

[76] 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.

[77] B. T. Loo, J. M. Hellerstein, R. Huebsch, S. Shenker, and I. Stoica. Enhancing P2P file-sharing with an internet-scale query processor. 2004.

[78] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein. The case for a hybrid P2P search infrastructure. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004.

[79] R. M. Losee. Comparing boolean and probabilistic information retrieval systems across queries and disciplines. *Journal of the American Society for Information Science*, 48(2):143–156, 1998.

[80] A. Löser, F. Naumann, W. Siberski, W. Nejdl, and U. Thaden. Semantic overlay clusters within super-peer networks. In *Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing in Conjunction with the VLDB 2003*, 2003.

[81] A. Löser, F. Naumann, W. Siberski, W. Nejdl, and U. Thaden. Semantic overlay clusters within super-peer networks. In *VLDB colocated Workshop: International Workshop on Databases, Information Systems, and P2P Computing*, 2003.

[82] J. B. Lovins. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11:22–31, 1968.

[83] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Survey and Tutorial*, 7(2), 2005.

[84] H. P. Luhn. The automatic creation of literature abstracts. *IBM Journal*, pages 159–165, 1958.

[85] H. P. Luhn. The automatic derivation of information retrieval encodements from machine-readable texts. *New York Interscience Publication*, pages 1021–1028, 1961.

[86] A. Medina, I. Matta, and J. Byers. On the origin of power laws in internet topologies. *ACM SIGCOMM Computer Communication Review*, 30(2), 2000.

[87] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical report, Hewlett-Packard Labs.

[88] C. N. Mooers. The theory of digital handling of non-numerical information and its implications to machine economics. *Technical Bulletin No. 48, Cambridge, MA.*

[89] W. Müller, M. Eisenhardt, and A. Henrich. Scalable summary based retrieval in p2p networks. Technical Report TR-MI-2005-01, Media Informatics, Bamberg University, 2005.

[90] 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 3rd International Semantic Web Conference (ISWC)*, 2004.

[91] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. EDUTELLA: a P2P networking infrastructure based on RDF. In *Proceedings of the eleventh World Wide Web Conference (WWW)*, pages 604–615, 2002.

[92] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmr, and T. Risch. EDUTELLA: a P2P Networking Infrastructure based on RDF. In *Proceedings of the Eleventh International World Wide Web Conference (WWW)*, Hawaii, USA, May 2002.

[93] W. S. Ng, B. C. Ooi, and K. L. Tan. BestPeer: A self-configurable peer-to-peer system. In *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, 2002.

[94] W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou. PeerDB: A P2P-based system for distributed data sharing. In *Proceedings of the 19th International Conference on Data Engineering (ICDE)*, 2003.

[95] A. Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies.* O'Reilly, 2001.

[96] C. D. Paice. Another stemmer. *ACM SIGIR Forum*, 24(3), 1990.

[97] W. W. Peterson. Adressing for random access storage. *IBM Journal Research and Development*, 1(2):130–146, 1957.

[98] C. Qu, W. Nejdl, and M. Kriesell. Cayley DHTs - a group-theoretic framework for analyzing DHTs based on cayley graphs. In *Proceedings of the 2nd International Symposium on Parallel and Distributed Processing and Applications (ISPA)*, 2004.

[99] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of the 2001 Conference on applications, technologies, architectures, and protocols for computer communications (ACM SIGCOMM).* ACM Press, 2001.

[100] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proceedings of NGC*, 2001.

[101] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, 2001.

[102] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. In *Proceedings of NGC*, 2001.

[103] G. Salton and M. E. Lesk. The SMART automatic document retrieval systems – an illustration. *Communcation of ACM*, 8(6):391–398, 1965.

[104] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, NY, 1988.

[105] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. HyperCuP—Hypercubes, Ontologies and Efficient Search on P2P Networks. In *International Workshop on Agents and Peer-to-Peer Computing*, Bologna, Italy, July 2002.

[106] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. Shaping up peer-to-peer networks. Technical Report TR-MI-2005-01, Stanford University, 2002.

[107] R. Schollmeier. A definition of peer-to-peer networking for the classification of peer-to- peer architectures and applications. In *Proceeding of the 1st International Conference on Peer-to-Peer Computing (P2P 2001)*, 2001.

[108] W. Siberski and U. Thaden. A simulation framework for schema-based query routing in P2P-networks. In *Proceedings of the 1st International Workshop on Peer-to-Peer Computing & DataBases(P2P& DB 2004*, 2004.

[109] M. Simkin and V. P. Roychowdhury. Read before you cite! *Complex Systems*, 14:269–274, 2003.

[110] M. P. Singh. Peering at peer-to-peer computing. *IEEE Internet Computing*, 5:4–5, 2001.

[111] F. Steimann, J. Gößner, and U. Thaden. Proposing mobile pair programming. *OOPSLA Workshop on Pair Programming.*

[112] F. Steimann, U. Thaden, W. Sibersk, and W. Nejdl. Animiertes UML als medium für die didaktik der objektorientierten programmierung. *Modellierung 2002*, 2002.

[113] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on applications, technologies, architectures, and protocols for computer communications (ACM SIGCOMM).* ACM Press, 2001.

[114] T. Suel, C. Mathur, J. Wu, J. Zhang, A. Delis, X. L. M. Kharrazi, and K. Shanmugasundaram. Odissea: A peer-to-peer architecture for scalable web search and information retrieval. *6th International Workshop on the Web and Databases (WebDB), June 2003*, 2003.

[115] C. Tang and S. Dwarkadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[116] C. Tang, Z. Xu, and M. Mahalingam. Peersearch: Efficient information retrieval in peer-peer networks. Technical Report HPL-2002-198, Hewlett-Packard Labs, 2002.

[117] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information retrieval in structured overlays. *ACM HotNets-I*, 2002.

[118] B. Tatomir, H. Dibowski, and L. Rothkrantz. Hierarchical routing in traffic networks.

[119] U. Thaden, T. Buchloh, and F. Steimann. iMOL]: Ein experimentelles Werkzeug zur interaktiven Validierung von Softwaremodellen, location = Marburg, year = 2004. In *Modellierung 2004*, GI Lecture Notes in Informatics.

[120] U. Thaden, W. Siberski, and W. Nejdl. A semantic web based peer-to-peer service registry network. Technical report, Learninglab Lower Saxony, 2003.

[121] U. Thaden and F. Steimann. Animated uml as a 3d-illustration for teaching oop. In *ECOOP: Seventh workshop on on Pedagogies and Tools for Learning Object-Oriented Concepts*, 2003.

[122] N. Ting and R. Deters. 3ls - a peer-to-peer network simulator. In *Third International Conference on Peer-to-Peer Computing*, 2003.

[123] P. Triantafillou and T. Pitoura. Towards a unifying framework for complex query processing over structured peer-to-peer data networks. In *Proceedings of the Workshop on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P, colocated VLDB workshop)*, 2003.

[124] C. L. Viles and J. C. French. Dissemination of collection wide information in a distributed information retrieval system. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 12–20. ACM Press, 1995.

[125] C. L. Viles and J. C. French. On the update of term weights in dynamic information retrieval systems. In *Proceedings of the 1995 International Conference on Information and Knowledge Management (CIKM)*, pages 167–174. ACM, 1995.

[126] S. Waterhouse. Jxta search: Distributed search for distributed networks, 2001. http://search.jxta.org/JXTAsearch.pdf.

[127] D. Winer and C. Shirky. Clay Shirky on P2P. http://davenet.scripting.com/-2000/11/15/clayShirkyOnP2p.

[128] S. Wray, T. Glauert, and A. Hopper. The medusa applications environment. In *International Conference on Multimedia Computing and Systems*, pages 265–273, 1994.

[129] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE)*, March 2003.

[130] W. G. Yee and O. Frieder. The design of PIRS, a peer-to-peer information retrieval system. *DBISP2P Workshop*, 2004.

[131] L. Zhou, R. van Renesse, and M. Marsh. Implementing IPv6 as a peer-to-peer overlay network. In *21st Symposium on Reliable Distributed Systems (SRDS)*, 2002.

[132] S. Q. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the Eleventh International Workshop on Network and OS Support for Digital Audio and Video.* ACM, 2001.

[133] J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. In *Proceedings of the ACM Transactions on Database Systems (TODS)*, pages 453–490, 1998.