

UNIVERSITY OF PISA



DEPARTMENT OF
INFORMATION ENGINEERING
DOCTORAL COURSE IN
INFORMATION ENGINEERING

MASARYK UNIVERSITY
FACULTY OF INFORMATICS



DEPARTMENT OF
INFORMATION TECHNOLOGIES
DOCTORAL COURSE IN
INFORMATICS

PH.D. THESIS

A Content-Addressable Network for Similarity Search in Metric Spaces

Fabrizio Falchi

SUPERVISOR
Lanfranco Lopriore

SUPERVISOR
Fausto Rabitti

SUPERVISOR
Pavel Zezula

2007

Abstract

Because of the ongoing digital data explosion, more advanced search paradigms than the traditional exact match are needed for content-based retrieval in huge and ever growing collections of data produced in application areas such as multimedia, molecular biology, marketing, computer-aided design and purchasing assistance. As the variety of data types is fast going towards creating a database utilized by people, the computer systems must be able to model human fundamental reasoning paradigms, which are naturally based on similarity. The ability to perceive similarities is crucial for recognition, classification, and learning, and it plays an important role in scientific discovery and creativity. Recently, the mathematical notion of metric space has become a useful abstraction of similarity and many similarity search indexes have been developed.

In this thesis, we accept the metric space similarity paradigm and concentrate on the scalability issues. By exploiting computer networks and applying the Peer-to-Peer communication paradigms, we build a structured network of computers able to process similarity queries in parallel. Since no centralized entities are used, such architectures are fully scalable. Specifically, we propose a Peer-to-Peer system for similarity search in metric spaces called Metric Content-Addressable Network (MCAN) which is an extension of the well known Content-Addressable Network (CAN) used for hash lookup. A prototype implementation of MCAN was tested on real-life datasets of image features, protein symbols, and text — observed results are reported. We also compared the performance of MCAN with three other, recently proposed, distributed data structures for similarity search in metric spaces.

Acknowledgments

It is a great honor and privilege for me to have an international joint supervision of my Ph.D. dissertation. I want to thank the University of Pisa and the Masaryk University, Brno for their agreement regarding this joint supervision. I am grateful to my supervisors Lanfranco Lopriore, Fausto Rabitti and Pavel Zezula for their guidance and encouragement. Since I was not just a Ph.D. student at the University of Pisa but also at the Masaryk University, I had the opportunity to work with several researchers in Brno, especially Michal Batko, David Novak and Vlastislav Dohnal. The discussions we had and the work we did together were fundamental for the completion of this thesis.

This thesis would not have been possible without the substantial help from Claudio Gennaro and without the discussions I had with Giuseppe Amato and Pasquale Savino. I also thank all the other people at the Networked Multimedia Information Systems laboratory of the Institute of Information Science and Technologies, especially Fausto Rabitti who is the head of the laboratory and his predecessor Costantino Thanos. I would also like to thank Cosimo Antonio Prete and Sandro Bartolini for teaching me the basics of research during my master thesis. Finally, I thank my mother and father for supporting me during my studies.

To Moly

Contents

Abstract	iii
Acknowledgments	v
Table of Contents	vii
Introduction	xiii
I.1 Similarity Search	xiv
I.2 Statement of the Problem	xv
I.2.1 Purpose of the Study	xvi
I.2.2 Significance of the Study	xvii
I.2.3 Limitations of the Study	xviii
I.2.4 Summary	xix
1 The Similarity Search Problem	1
1.1 Similarity Search	1
1.2 Similarity Queries	3
1.2.1 Range Query	4
1.2.2 Nearest Neighbor Query	4
1.2.3 Combinations of Queries	5
1.2.4 Complex Similarity Queries	5
1.3 Metric Spaces	7
1.3.1 Metric Distance Measures	9
1.4 Access Methods for Metric Spaces	12
1.4.1 Ball Partitioning	12
1.4.2 Generalized Hyperplane Partitioning	12
1.4.3 Exploiting Pre-Computed Distances	13

1.4.4	Hybrid Indexing Approaches	13
2	Distributed Indexes	15
2.1	Scalable and Distr. Data Structures	16
2.2	Peer-to-Peer Systems	19
2.2.1	Characterization	21
2.2.2	The lookup problem	25
2.3	Unstructured Peer-to-Peer Systems	27
2.4	Structured Peer-to-Peer Systems	29
2.4.1	Introduction to DHTs	29
2.4.2	Chord	36
2.4.3	Pastry	39
2.4.4	Tapestry	41
2.4.5	Chimera	42
2.4.6	Z-Ring	42
2.4.7	Content Addressable Network <i>CAN</i>	43
2.4.8	Kademlia	43
2.4.9	Symphony	46
2.4.10	Viceroy	47
2.4.11	DHTs Comparison	49
2.4.12	DHTs Related Works	49
2.4.13	P-Grid	50
2.4.14	Small-World and Scale-Free	51
2.4.15	Other Works	52
2.5	Metric Peer-to-Peer Structures	52
2.5.1	GHT* and VPT*	53
2.5.2	M-Chord	57
2.6	Peer-to-Peer and Grid Computing	59
3	Content-Addressable Network (CAN)	63
3.1	Node arrivals	64
3.1.1	Finding a Zone	66
3.1.2	Joining the Routing	67
3.2	Routing	67
3.3	Node departures	69
3.4	Load Balancing	70

3.5	M-CAN: CAN-based Multicast	70
3.6	CAN Related Works	74
4	MCAN	77
4.1	Mapping	78
4.1.1	Pivot Selection	79
4.2	Filtering	80
4.3	Regions	81
4.4	Construction	82
4.5	Insert	83
4.6	Split	84
4.7	Execution End Detection	85
4.8	Range Query	86
4.9	Nearest Neighbor query	88
5	MCAN Evaluation	95
5.1	Dimensionality of the Mapped Space	95
5.2	Range query	98
5.3	Nearest Neighbor query	103
5.3.1	Number of peers involved in query execution	106
5.3.2	Total number of distance computations . . .	107
5.3.3	Parallel cost of kNN	112
5.3.4	Candidate results	112
6	MCAN Comparison with other structures	117
6.1	Experiments Settings	118
6.2	Measurements	119
6.3	Scalability with Respect to the Size of the Query .	120
6.4	Scalability with Respect to the Size of Datasets . .	130
6.5	Number of Simultaneous Queries	136
6.6	Comparison Summary	140
7	Conclusions	143
7.1	Research Directions	144
	Bibliography	145

Introduction

Traditionally, search has been applied to structured (attribute-type) data yielding records that exactly match the query. However, to find images similar to a given one, time series with similar development, or groups of customers with common buying patterns in respective data collections, the traditional search technologies simply fail. In these cases, the required comparison is gradual, rather than binary, because once a reference (query) pattern is given, each instance in a search file and the pattern are in a certain relation which is measured by a user-defined dissimilarity function.

These types of search are more and more important for a variety of current complex digital data collections and are generally designated as similarity search. Thus similarity search has become a fundamental computational task in many applications. Unfortunately, its cost is still high and grows linearly with respect to the dataset size which prevents the realization of efficient applications for the ever-increasing amount of digital information being created today.

In the last few years, Peer-to-Peer systems have been widely used particularly for file-sharing, distributed computing, Internet-based telephony and video multicast. The growth in the usage of these applications *“is enormous and even more rapid than that of the World Wide Web”* [183]. In a Peer-to-Peer system autonomous entities (peers) aim for the shared usage of distributed autonomous resources in a networked environment. Taking advantage of these distributed resources, scalability issues of centralized solutions for specific applications have been overcome.

In this thesis we present a scalable distributed similarity search

structure for similarity search in metric spaces, which is called Metric Content-Addressable Network (MCAN). MCAN extends the well known structured Peer-to-Peer system CAN described in Chapter 3 [p.63], to generic metric space objects. MCAN uses the Peer-to-Peer paradigm to overcome scalability issues of centralized structures for similarity search, taking advantage of distributed resources over the Peer-to-Peer network.

I.1 Similarity Search

Similarity search is based on gradual rather than exact relevance and is used in content-based retrieval for queries involving complex data types such as images, videos, time series, text and DNA sequences. For instance, for the experimental results reported in this thesis, we use 3 datasets which cover very different topics: vectors of color image features, protein symbol sequences, and titles and subtitles of books and periodicals from several academic libraries.

In similarity search, discussed in Chapter 1 [p.1], a distance between objects is used to quantify the *proximity*, *similarity* or *dissimilarity* of a query object versus the objects stored in a database to be searched. Although many similarity search approaches have been proposed, the most generic one considers the mathematical notion of metric space (see Section 1.3 [p.7]) as a suitable abstraction of similarity. The simple but powerful concept of the metric space consists of a domain of objects and a distance function which satisfies a set of constraints. It can be applied not only to multi-dimensional vector spaces, but also to different forms of string objects, as well as to sets or groups of various natures, etc. However, the similarity search is inherently expensive and for centralized indexes the search costs increase linearly as the stored dataset grows, thus preventing them from being applied to huge files that have become common and are continuously growing.

Applications of similarity search for multimedia objects have been studied extensively. In the last few years, standardization processes have taken place to make multimedia features interchangeable,

thus moving the problem of comparing two multimedia objects to the problem of comparing their standardized features which could be automatically extracted by different software. In particular in the Multimedia Content Description Interface standard MPEG-7 [87], a great number of low level features for multimedia objects have been defined. Moreover, distance functions to evaluate the dissimilarity of two multimedia objects have been suggested by the Moving Picture Experts Group (MPEG) [154]. The low level features of MPEG-7 have been largely used in advanced multimedia information systems (e.g. MILOS Photo Book [9, 10]).

I.2 Statement of the Problem

Current centralized similarity search structures for metric spaces reveal linear scalability with respect to the data set size. Obviously, this is not sufficient for the expected data volume dimension of the problem, and this is why there is no web search engine able to search the web for images similar to a given one. In fact, the image search application available (e.g. Google Image Search¹, Yahoo! Image Search²) are simply based on the accompanying or manual annotated text (e.g. Google Image Labeler³).

Peer-to-Peer systems are today widely used to perform unscalable tasks in network of volunteer nodes (e.g. file sharing, search for extraterrestrial intelligence, video broadcasting over the Internet, etc.). However, centralized indexes for exact match queries are still largely used in file sharing communities. In this context, distributed indexes are mostly used because it is more difficult for the authorities to shut down them. Distributed Peer-to-Peer web search engines have also been proposed to avoid the existence of a centralized control (e.g. Google).

In recent years various structured Peer-to-Peer systems have been proposed for exact match indexing (see Section 2.4 [p.29]).

¹<http://images.google.com/>

²<http://images.search.yahoo.com/>

³<http://images.google.com/imagelabeler/>

However, their use is still limited because of the good performance achievable by the centralized structures. We believe that structured Peer-to-Peer systems for similarity search are more attractive because their centralized counterparts do not scale well and thus they can not be used for huge amount of data.

Very recently four scalable distributed similarity search structures for metric data have been proposed. One is the MCAN which is the object of this thesis. The other three are GHT*, VPT* and M-Chord which we describe in Section 2.5 [p.52]. Each of these structures is able to execute similarity queries for any metric dataset, and they all exploit parallelism for query execution. All these distributed similarity search structures adopt the Peer-to-Peer paradigm trying to overcome scalability issues of similarity search structures using distributed resources over a Peer-to-Peer network.

1.2.1 Purpose of the Study

In this thesis we consider the Peer-to-Peer paradigm to develop a scalable and distributed data structure for similarity search in metric spaces. Structured Peer-to-Peer systems have been proposed to efficiently find data in a scalable manner in large Peer-to-Peer systems using exact match. Almost all of them belong to the class of Distributed Hash Tables (DHTs) discussed in Section 2.4 [p.29].

Our objective is to study the possibility of combining the capabilities of metric space similarity search with the power and high scalability of the Peer-to-Peer systems, especially DHTs, employing their virtually unlimited storage and computational resources. For this purpose we extend the Content-Addressable Network (CAN), which is a well-known DHT, to support storage and retrieval of generic metric space objects. More precisely, we want to address the problem of executing Range and Nearest Neighbor queries (see Section 1.2 [p.3]). Particular attention is given to the scalability of the proposed solution with respect to the dataset size.

Because of the great variety of forms of data that do satisfy the metric assumptions, we evaluate the performance of the proposed solution using different datasets (i.e. of image features, protein

symbols and text) and different metric distance functions.

Considering that almost at the same time that we were developing our MCAN, other three similar structures (i.e. GHT*, VPT* and M-Chord) were proposed, we want to compare their performance considering various performance measures and various datasets.

I.2.2 Significance of the Study

The proposed MCAN has been proved to be scalable with respect to the dataset size considering the response time of Range queries. Three different strategies for performing Nearest Neighbor queries were studied. The study demonstrated that the mixed mode execution is the best choice in general. It responds well to the demand of the scalability in terms of response time and consumes fewer resources than the complete parallel approaches.

Implementing MCAN over the same framework in which, almost at the same time, three similar structures (i.e. GHT*, VPT* and M-Chord) were built, it was possible to explore the performance characteristics of the proposed algorithms by showing their advantages and disadvantages.

The distributed similarity search structure approach is the core of the European project SAPIR⁴ (Search on Audio-visual content using Peer-to-peer Information Retrieval) that aims at finding new ways to analyze, index, and retrieve the tremendous amounts of speech, image, video, and music that are filling our digital universe, going beyond what the most popular engines are still doing, that is, searching using text tags that have been associated with multimedia files. SAPIR is a three-year research project that aims at breaking this technological barrier by developing a large-scale, distributed Peer-to-Peer architecture that will make it possible to search for audio-visual content by querying the specific characteristics (i.e. features) of the content. SAPIR's goal is to establish a giant Peer-to-Peer network, where users are peers that produce audiovisual

⁴<http://sysrun.haifa.il.ibm.com/sapir/>

content using multiple devices and service providers are super-peers that maintain indexes and provide search capabilities.

MCAN will be also used in the context of the Networked Peers for Business (NeP4B)⁵, an Italian FIRB project which aims to contribute innovative Information and Communication Technologies solutions for small and medium enterprises by developing an advanced technological infrastructure to enable companies of any nature, size and geographic location to search for partners, exchange data, negotiate and collaborate without limitations and constraints.

1.2.3 Limitations of the Study

MCAN efficiently supports similarity search of a single type of feature. However, even for the same type of data, different features can be used for searching. For instance, in content based image retrieval parts of the entire images, obtained using image segmentation software, can be searched considering different aspects, e.g. the colors, the shape, the edges and the texture. Moreover, in most cases a combination of features is requested. Whenever the combination is fixed and the combined distance function is metric, our proposed solution could be used, considering the combined features as a whole. However, in some cases, the users would like to dynamically specify the features to combine, the importance to give to each feature and the combination function. To this purpose, the state of the art complex queries algorithms that should be used (see Subsection 1.2.4 [p.5]), typically require an Incremental Nearest Neighbor algorithm to be implemented for each feature. Moving from the Nearest Neighbor algorithm presented in this thesis, it should be possible to define an Incremental version. However, this is still an open question which will be investigated in the near future.

Using CAN as the basis, MCAN inherited a lot of functionalities that were proposed in the literature for the CAN (e.g. load balancing, multicast, recovering from node failures, etc.). However, the performance of these functionalities should be extensively studied.

⁵<http://www.dbgroup.unimo.it/nep4b/>

With distributed data structures for similarity search as the MCAN, we achieved scalability of similarity queries with respect to the dataset size adding resources as the dataset grows. Basically, we supposed that resources (i.e. nodes) are at least proportional to the dataset size which is quite common in Peer-to-Peer systems used by communities. However, similarity queries do remain expensive tasks which require a lot of resources.

I.2.4 Summary

The thesis is organized as follows. In Chapter 1 backgrounds and definitions of the similarity search problem are given, with particular attention to the metric space abstraction. Scalable indexing mechanisms in distributed systems are the topic of Chapter 2 where the Scalable and Distributed Data Structures (SDDSs) and Peer-to-Peer approaches are discussed. The Content-Addressable Network, which is the base of our structure, is then discussed in Chapter 3. Our Metric Content-Addressable Network, which is our Peer-to-Peer similarity search structure, is defined in Chapter 4. Exhaustive experimental evaluation is provided in Chapter 5. Finally, the results of a comprehensive experimental comparison between our and other three scalable distributed similarity search structures are reported in Chapter 6.

Chapter 1

The Similarity Search Problem

Similarity search, particularly in metric spaces, has been receiving increasing attention in the last decade, due to its many applications in widely different areas. Two recent books [191, 157], two surveys in ACM Computing Surveys (CSUR) [46, 29] and ACM Transactions on Database Systems (TODS) [84, 30], and the sharp increase in publications in recent years witness the momentum gained by this important problem.

The chapter is organized as follows. In Section 1.1 we discuss the notion of similarity and we define the search problem. In Section 1.2 we will report the most used similarity queries. The metric space abstraction is presented in Section 1.3 together with some metric distance measures. Finally, we illustrate the most important centralized access method for metric spaces that has been presented in the literature.

1.1 Similarity Search

The notion of similarity has been studied extensively in the field of psychology and the given definition characterizes that *similarity* has been found to have an important role in cognitive sciences. In [75], R.L. Goldstone and S.J. Yun say:

An ability to assess similarity lies close to the core of cognition. In the time-honored tradition of legitimizing fields of psychology by citing William James, “This sense of Sameness is the very keel and backbone of our thinking” [89, p.459]. An understanding of problem solving, categorization, memory retrieval, inductive reasoning, and other cognitive processes requires that we understand how humans assess similarity. Four major psychological models of similarity are geometric, featural, alignment-based, and transformational.

From a database prospective, similarity search is based on gradual rather than exact relevance. A distance between objects is used to quantify the *proximity*, *similarity* or *dissimilarity* of a query object versus the objects stored in a database to be searched.

A similarity search can be seen as a process of obtaining data objects in order of their distance or dissimilarity from a given query object. It is a kind of *sorting*, *ordering*, or *ranking* of objects with respect to the query object, where the ranking criterion is the distance measure. Though this principle works for any distance measure, we restrict the possible set of measure to the *metric* distance (see Section 1.3 [p.7]). Because of the mathematical foundations of the *metric space* notion, partitioning and pruning rules can be constructed for developing efficient index structures. Therefore, in the past years research has focused on *metric spaces*.

On the other hand, since many data domains in use are represented by vectors, we could restrict to this special case of *coordinate spaces* which is a special case of metric space in which objects can be seen as vectors. Unfortunately existing solutions for searching in coordinate space suffer from the so-called *dimensionality curse*, i.e. either become slower than naive algorithms with linear search times or they use too much space. Moreover, some spaces have coordinates restricted to small sets of values, so that the use of such coordinates could not be helpful. Therefore, even if the use of coordinates can be advantageous in special cases, as Clarkson said in [53]:

to strip the problem down to its essentials by only considering distances, it is reasonable to find the minimal properties needed for fast algorithms.

As in [191], we consider the problem of organizing and searching large datasets from the perspective of *generic* or *arbitrary* metric space, sometimes conveniently labelled *distance space*. In general, the search problem can be described as follows:

Problem 1. Let \mathcal{D} be the domain, d a distance measure on \mathcal{D} , and $\mathcal{M} = (\mathcal{D}, d)$ a metric space. Given a set $\mathcal{X} \subseteq \mathcal{D}$ of elements, preprocess or structure the data so that proximity queries are answered efficiently.

From a practical point of view, \mathcal{X} can be seen as a file (a dataset or a collection) of objects that takes values from the domain \mathcal{D} , with d as the proximity measure, i.e. the distance function defined for an arbitrary pair of objects from \mathcal{D} . In (Section 1.2 [p.3]) the most common similarity queries will be presented.

In this thesis we will only considering metric distance functions (see Section 1.3 [p.7]). The most important ones are presented in Subsection 1.3.1 [p.9].

1.2 Similarity Queries

A *similarity query* is defined by a query object and a constraint on the proximity required for an object to be in the result set. The response to a query returns all the objects that satisfy the selection conditions.

In this section we present the two basic types of similarity queries, i.e. *Range* and *Nearest Neighbor*. We also consider the *Incremental Nearest Neighbor search* and the problematics of combining similarity queries and of complex similarity queries. For a more complete description of similarity queries see [191, ch. 4].

1.2.1 Range Query

The *similarity range query* $R(q,r)$ is probably the most common type of similarity search. The query is specified by a query object $q \in \mathcal{D}$, with some query radius $r \in \mathbb{R}$ as the distance constraint. The query retrieves all the objects within distance r of q , formally:

$$R(q,r) = \{x \in \mathcal{X} \subseteq \mathcal{D}, d(x,q) \leq r\} .$$

If need, individual objects in the response set can be ranked according to their distance with respect to q . Observe that the query object q needs not exist in the collection $\mathcal{X} \subseteq \mathcal{D}$ to be searched. The only restriction on q is that it belongs to the domain \mathcal{D} . When the range radius is zero, the range query $R(q,r)$ is called a *point query* or *exact match*. In this case, we are looking for an identical copy (or copies) of the query object q . The most usual use of this type of query is in delete algorithms, when we want to locate an object to remove it from the database.

1.2.2 Nearest Neighbor Query

Whenever we want to search for similar objects using a range search, we must specify a maximal distance for objects to qualify. But it can be difficult to specify the radius without some knowledge of the data and the distance function. For example, the range $r = 3$ of the edit distance metric represents less than four edit operations between compared strings. This has a clear semantic meaning. However, a distance of two color-histogram vectors of images is a real number whose quantification cannot be so easily interpreted. If too small a query radius is specified, the empty set may be returned and a new search with a larger radius will be needed to get any result. On the other hand, if query radii are too large, the query may be computationally expensive and the response sets contain many non significant objects.

An alternative way to search for similar objects is to use Nearest Neighbor query. The elementary version of this query finds the closest object to the given query object, that is the nearest neigh-

bor of q . The concept can be generalized to the case where we look for the k nearest neighbors. Specifically $kNN(q)$ query retrieves the k nearest neighbors of the object q . If the collection to be searched consists of fewer than k objects, the query returns the whole database. Formally, the response set can be defined as follows:

$$kNN(q) = \{\mathcal{R} \subseteq \mathcal{X}, |\mathcal{R}| = k \wedge \forall x \in \mathcal{R}, y \in \mathcal{X} - \mathcal{R} : d(q, x) \leq d(q, y)\} .$$

When several objects lie at the same distance from the k -th nearest neighbor, the ties are solved arbitrarily.

1.2.3 Combinations of Queries

As an extension of the query types defined above, we can define additional types of queries as combinations of the previous ones. For example we might combine a range query with a nearest neighbor query to get $kNN(q, r)$ with the response set:

$$kNN(q, r) = \{\mathcal{R} \subseteq \mathcal{X}, |\mathcal{R}| \leq k \wedge \forall x \in \mathcal{R}, y \in \mathcal{X} - \mathcal{R} : d(q, x) \leq d(q, y) \wedge d(q, x) \leq r\} .$$

In fact, we have constrained the result from two sides. First, all objects in the result-set should lie at a distance not greater than r , and if there are more than k of them, just the first (i.e. the nearest) k are returned.

1.2.4 Complex Similarity Queries

Complex similarity queries are queries consisting of more than one similarity predicate. Efficient processing of such kind of queries differs substantially from traditional (Boolean) query processing. The problem was studied first by Fagin in [60] (related works are

[61, 62, 63, 64]. The similarity score (or grade) a retrieved object receives as a whole depends not only on the scores it gets for individual predicates, but also on how such score are combined.

To this aim, Fagin has proposed the \mathcal{A}_0 algorithm [60]. This algorithm assumes that for each query predicate we have an index structure able to return objects in order of decreasing similarity. Other related papers are [78, 134].

Ciaccia, Patella, and Zezula [51] have concentrated on complex similarity queries expressed through a generic language. On the other hand, they assume that query predicates are from a single feature domain, i.e. from the same metric space. The proposed evaluation process is based on distances between feature values, because metric indexes can use just distances to evaluate predicates. In [49] an extension of M-tree [50] is given. It outperforms the \mathcal{A}_0 algorithm but has the main drawback is that even though it is able to employ more features during the search, these features are compared using a single distance function.

A generalization of relational algebra to allow the formulation of complex similarity queries over multimedia database called *similarity algebra with weights* has been introduced in [48].

Incremental Nearest Neighbor Search

When finding k nearest neighbors to the query object using a kNN algorithm (see Section 1.2.2 [p.4]), k is known prior to the invocation of the algorithm. Thus if the $(k+1)$ -th neighbor is needed, the kNN needs to be reinvoked for $(k+1)$ neighbors from scratch. To resolve this problem, the authors of the *Incremental Nearest Neighbor* algorithm [83] proposed the concept of *distance browsing* which is to obtain the neighbors incrementally (i.e. one by one) as they are needed. This operation means browsing through the database on the basis of distance.

An *incremental similarity search* can provide objects in order of decreasing similarity without explicitly specifying the number of nearest neighbors in advance. This is especially important in interactive database applications, as it makes it possible to display

partial query results early. For more detail see [32] and [83]. The incremental aspect also provides significant benefits in situations where the number of desired neighbors is unknown beforehand, for example when complex similarity queries are processed (see [62]).

1.3 Metric Spaces

Although many similarity search approaches have been proposed, the most generic one considers the mathematical metric space as a suitable abstraction of similarity.

Let $\mathcal{M} = (\mathcal{D}, d)$ be a metric space defined for a domain of objects (or the objects' *keys* or *indexed feature*), \mathcal{D} and a total (distance) function d . In this metric space, the properties of the function $d : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}$, sometimes called the metric space postulates, are typically characterized as:

$$\begin{array}{ll}
 \forall x, y \in \mathcal{D}, d(x, y) \geq 0 & \text{non-negativity,} \\
 \forall x, y \in \mathcal{D}, d(x, y) = d(y, x) & \text{symmetry,} \\
 \forall x, y \in \mathcal{D}, x = y \Leftrightarrow d(x, y) = 0 & \text{identity,} \\
 \forall x, y, z \in \mathcal{D}, d(x, z) \leq d(x, y) + d(y, z) & \text{triangle inequality.}
 \end{array}$$

$d(x, y)$ is called the *distance* from x to y . A function d satisfying the metric postulates is called a *metric function* [96] or simply the *metric*. Therefore, if d is a metric function can be called *metric distance*.

There are also several variations of metric spaces. In order to specify them more easily, we first transform the metric space postulates given above, into an equivalent form in which the identity postulate is decomposed into *reflexivity* (1.3) and *positiveness* (1.4):

$$\forall x, y \in \mathcal{D}, d(x, y) \geq 0 \quad \text{non-negativity,} \quad (1.1)$$

$$\forall x, y \in \mathcal{D}, d(x, y) = d(y, x) \quad \text{symmetry,} \quad (1.2)$$

$$\forall x \in \mathcal{D}, d(x, x) = 0 \quad \text{reflexivity,} \quad (1.3)$$

$$\forall x, y \in \mathcal{D}, x \neq y \Rightarrow d(x, y) > 0 \quad \text{positiveness,} \quad (1.4)$$

$$\forall x, y, z \in \mathcal{D}, d(x, z) \leq d(x, y) + d(y, z) \quad \text{triangle inequality.} \quad (1.5)$$

If the total (distance) function d does not satisfy the positive-ness property (1.4), it is called a *pseudo-metric* [96]. Although for simplicity we do not consider pseudo-metric spaces in this work, such functions can be transformed to the standard metric by regarding any pair of objects with zero distance as a single object. In fact if d is a pseudo-metric function, using triangle inequality (1.5):

$$d(x, y) = 0 \Rightarrow \forall z \in \mathcal{D}, d(x, z) = d(y, z) .$$

If the symmetry property (1.2) does not hold, we talk about a *quasi-metric*. For example, let the objects be different locations within a city, and the distance function the physical distance a car must travel between them. The existence of one-way streets implies the function must be asymmetrical. There are techniques to transform asymmetric distances into symmetric form, for example:

$$d_{sym}(x, y) = d_{asym}(x, y) + d_{asym}(y, x) .$$

If the function d satisfies a stronger constraint on the triangle equality:

$$\forall x, y, z \in \mathcal{D}, d(x, z) \leq \max\{d(x, y), d(y, z)\} .$$

d is called *super-metric* or *ultra-metric*. The geometric characterization of the super-metric requires every triangle to have at least two sides of equal length, i.e. to be *isosceles*, which implies that the third side must be shorter than the others. Ultra-metrics are widely used in the field of biology, particularly in evolutionary biology. By comparing the DNA sequences of pairs of species,

evolutionary biologists obtain an estimate of the time which has elapsed since the species separated. From these distances, an evolutionary tree (sometimes called phylogenetic tree) can be reconstructed, where the weights of the tree edges are determined by the time elapsed between two speciation events [142, 143]. Having a set of extant species, the evolutionary tree forms an ultra-metric tree with all the species stored in leaves and an identical distance from root to leaves. The ultra-metric tree is a model of the underlying ultra-metric distance function.

1.3.1 Metric Distance Measures

In the following, we present examples of distance functions used in practice on various types of data. Typically, distance functions are specified by domain experts. Depending on the character of values returned, distance measures can be divided into two groups: discrete (i.e. distance functions that return only a predefined small set of values) and continuous (i.e. distance functions in which the cardinality of the set of values returned is very large or infinite).

A survey of the most important metric distances can be found in [191, ch. 3].

Minkowski Distances

The Minkowski distance functions form a whole family of metric functions. They are usually designated as the L_p metrics where p is a parameter. These functions are defined on n -dimensional vectors of real numbers as:

$$L_p [(x_1, \dots, x_n), (y_1, \dots, y_n)] = \sqrt[p]{\sum_{i=1}^n |x_i - y_i|^p} .$$

The L_1 distance is known as the *Manhattan distance* (also called the *City-Block distance*). The L_2 distance denotes the well-known *Euclidean distance*. The *maximum distance* or *infinite distance* or *chessboard distance* is defined as:

$$L_\infty = \max_{i=1}^n |x_i - y_i| .$$

The L_p metrics are used in measurements of scientific experiments, environmental observations, or the study of different aspects of the business process.

Quadratic Form Distance

The *quadratic form distance function* has been successfully applied to vectors that have individual components correlated, i.e. a kind of cross-talk exists between individual dimensions (e.g. color histograms of images [67, 80, 158]).

A generalized quadratic distance measure is defined as

$$d_M(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T \cdot M \cdot (\vec{x} - \vec{y})} ,$$

where M is a semi-definite matrix where the *weights* $m_{i,j}$ denote how strong the connection between two components i and j of vectors \vec{x} and \vec{y} is.

Observe that this definition also subsumes the Euclidean distance when M is the identity matrix. When $M = \text{diag}(w_1, \dots, w_n)$ the quadratic form distance collapse to the so called *weighted Euclidean distance*:

$$d_M(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^n w_i (x_i - y_i)^2} .$$

Edit Distance

The closeness of sequences of symbols (strings) can be effectively measured by the *edit distance*, also called the *Levenshtein distance* presented in [101]. The distance between two string x and y is defined as the minimum number of atomic edit operations (insert, delete and replace) needed to transform string x into string y .

The generalized edit distance function assigns weights (positive real numbers) to individual atomic operations. Hence, the generalized edit distance is the minimum value of the sum of weighted atomic operations needed to transform x into y . In case the weight of insert and delete operations differ the generalized edit distance is not metric.

Using weighting functions, we can define a most generic edit distance which assigns different costs even to operations on individual characters. In this case, to retain the metric postulates some additional limits must be placed on weight functions (e.g. symmetry of substitutions).

A survey on string matching can be found in [131].

Tree Edit Distance

A proximity measure for trees is the *tree edit distance* (see [77, 117]) which defines a distance between two tree structures as the minimum cost needed to convert the source tree to the target tree using a predefined set of tree edit operation, such as the insertion or deletion of a node (i.e. a generalization of the edit distance to labeled trees). Since XML documents are typically modeled as rooted labeled trees, the tree edit distance can also be used to measure the structural dissimilarity of XML documents.

Jacard's Coefficient

Assuming two sets A and B , *Jacard's coefficient* is defined as follows:

$$d(A, B) = 1 - \frac{A \cap B}{A \cup B}.$$

An example of an application is measuring similarity of users search interest considering the URLs the accessed browsing the Internet. An application of this metric to vector data is called the *Tanimoto similarity measure*. An even more complicated distance measure defined on sets is the *Hausdorff distance* [86].

1.4 Access Methods for Metric Spaces

Relevant surveys on indexing techniques in metric spaces have been published by Chávez, Navarro, Baeza-Yates, and Marroquín in 2001 [46], by Hjaltason and Samet in 2003 [84] and by Zezula, Amato, Dohnal, and Batko in 2006 [191, ch. 2].

As in [191] we divide the individual techniques into four groups: Ball Partitioning, Generalized Hyperplane Partitioning, Exploiting Pre-Computed Distances, Hybrid Indexing Approaches. We will not take into account techniques for approximate similarity search. For a survey of approximate similarity search see [191, ch. 2.5]).

1.4.1 Ball Partitioning

Ball partitioning requires only one pivot and, provided the median distance is known, the resulting subsets contain the same amount of data. Numerous indexing approaches based on this simple concept have being defined.

The most important techniques using ball partitioning are: Burkhard-Keller Tree (BKT) [34], Fixed Queries Tree (FQT) [14], Fixed Queries Array (FQA) [45], Vantage Point Tree (VPT) [188], Excluded middle Vantage Point Forest [189].

1.4.2 Generalized Hyperplane Partitioning

An orthogonal approach to ball partitioning is Generalized Hyperplane Partitioning. Two reference objects (pivots) are arbitrarily chosen and assigned to two distinct objects subset. All objects are assigned to the subset containing the pivot which is nearest to the object itself. In contrast to ball partitioning the generalized hyperplane does not guarantee a balanced split, and a suitable choice of reference points to achieve this objective is an interesting challenge.

The most important techniques of this type are: Bisector Tree (BST) [92], Generalized Hyperplane Tree (GHT) [176].

1.4.3 Exploiting Pre-Computed Distances

In [160] Shasha and Wang suggested using pre-computed distances between data objects. Pairwise distance which are not stored (typically those from a query object to database objects) are estimated as intervals using the pre-computed distances. This technique of storing and using pre-computed distances may be effective for datasets of small cardinality, but space requirements and search complexity become overwhelming for larger files.

The most important techniques of this type are: Approximating and Eliminating Search Algorithm (AESA) [177, 178], Linear AESA (LAESA) [122] Spaghettis [44]. LAESA and Spaghettis, in particular, store distances from objects to only a fixed number of pivots chosen with specific algorithms.

1.4.4 Hybrid Indexing Approaches

Indexing methods that employ pre-computed distances provide high performance boosts in terms of computational costs. However, the price to pay for their performance is their enormous space requirements. A straightforward remedy is to combine both the partitioning principle and the pre-computed distances technique into a single index structure.

The most important hybrid indexing approaches are: Multi Vantage Point Tree (MVPT) [30], Geometric Near-neighbor Access Tree(GNAT) [31], Spatial Approximation Tree (SAT) [130, 132], Metric Tree (M-tree) [50] (see [191, ch. 3] for a survey of M-Trees variants), Similarity Hashing (SH) [73], D-Index [57].

Chapter 2

Distributed Indexes

During the last few years, numerous papers have been published about scalable indexes in distributed environments. As Aberer suggested in [3], two classes have been studied in the literature: *Scalable and Distributed Data Structures* (SDDSs) and *Distributed Hash Tables* (DHTs).

SDDSs have been investigated for indexing scalable and distributed databases on workstation clusters. In most of the cases, they are variants of distributed search trees and hash-based access structures. SDDSs are characterized by a client-server architecture, a medium number of nodes, and typical by some form of global coordination, such as split coordinators or global directories. SDDSs are analyzed in more details in Section 2.1.

DHTs, which represent the most important class of structured Peer-to-Peer systems, have been studied for implementing global-scale resource access in Peer-to-Peer architectures. DHTs differ from SDDSs through complete (or almost complete) degree of decentralization. They implement routing schemes for quickly locating resources identified by a data key in a network of N peers (typically in $O(\log(N))$ time). The search can be started at any peer, without relying on a centralized directory. In Section 2.2 an introduction to the Peer-to-Peer paradigm is given while unstructured and structured Peer-to-Peer systems are discussed in Section 2.3 and Section 2.4 respectively. Finally in Section 2.5 we illustrate

other 3 distributed similarity search structures that were developed at the same time we developed our MCAN.

2.1 Scalable and Distr. Data Structures

In 1993 the use of mass produced PCs and WSs, interconnected through high speed networks (10 Mb/s - 1 Gb/s) were becoming prevalent in many organizations. Multicomputers needed new system software, fully taking advantage of the distributed RAM and of parallel processing on multiple CPUs. As Litwin, Neimat, and Schneider said in [113], a frequent problem of developing such kind of software is that:

[...] while the use of too many sites may deteriorate performance, the best number of sites to use is either unknown in advance or could evolve during processing. Given a client/server architecture, we are interested in methods for gracefully adjusting the number of servers, i.e. the number of sites or processors involved.

Basically, the problem was to find data structures that efficiently would use the servers.

Litwin, Neimat, and Schneider in [111, 113] defined Scalable Distributed Data Structure (SDDS) to solve this problem. They defined a SDDS as a structure that meets tree constraints:

1. A file expands to new servers gracefully, and only when servers already used are efficiently loaded.
2. There is no master site that object address computations must go through, e.g. to access a centralized directory.
3. The file access and maintenance primitives, e.g. search, insertion, split, etc., never require atomic updates to multiple clients.

As Zezula, Amato, Dohnal, and Batko in [191], we refer these tree constraints respectively as: *scalability*, *no hot-spot* and *independence*.

In [112] the same authors gave a slightly different definition of an SDDS:

An SDDS stores data on some sites called server, and is used from some sites called clients, basically distinct, but not necessarily. [...] Every SDDS must respect three design requirements.

- 1. First, no central directory is used for data addressing, to avoid a hot-spot.*
- 2. Next, a client image can be outdated, being updated only through messages, called Image Adjustment Messages (IAMs). These are sent only when a client makes an addressing error. A client autonomy is preserved in this way, which is a major requirement for multicomputers [126].*
- 3. Finally, a client with an outdated image can send a key to an incorrect server, in which case the structure should deliver it to the right server, and trigger an IAM.*

While the *no hot-spot* and *independence* properties, as they are called in [191], are present in both the definitions (i.e. [111, 113] and [112]), the *scalability* property is here missed. Obviously SDDS are still considered scalable (it's in the name), but the way in which the scalability is achieved is no more specified. Instead, in [112], the authors gave a further specification of the message behavior. Essentially an SDDS must be tolerant to client addressing errors. This property was already present in the LH* [111], but it was not considered in the SDDS definition.

From the very beginning it was clear that there are two important goals in SDDSs: minimizing the messages and maximizing the *load factor* load factor.

As Litwin, Neimat, and Schneider said in [111],

to make an SDDS efficient, one should minimize the messages exchanged through the network, while maximizing the load factor.

The SDDSs studied in the literature can be grouped in four classes:

- *SDDSs for hashed files.*

Distributed Linear Hashing (LH*) [111, 56, 113, 109, 105, 179, 93].

They extend the more traditional dynamic hash data structures, especially the popular linear hashing [106, 107, 159] used, e.g. in the Netscape browser, and the dynamic hashing [99], to the network multicomputers, and switched multicomputers.

- *SDDSs for ordered files.*

RP* (a family of Order Preserving SDDS) [112], Distributed Random Tree (DRT) [97].

They extend the traditional ordered data structures, B-trees or binary trees, to the multicomputers.

- *SDDSs for multi-attribute (multi-key) files.*

k-RP*S [110], [129].

These algorithms extend the traditional k-d data structures [156] to the multicomputers. Performance of multi-attribute search may get improved by several orders of magnitude.

- *high-availability SDDSs.*

LH*s [105, 109].

They have no known counterpart among the traditional data structures. They are designed to transparently survive failures of server sites. They apply principles of mirroring, or of striping, or of grouping of records, revised to support the file scalability.

A number of systems have been built using SDDSs, by Litwin et al. (see [108]), as well as the P-Grid system at EPFL in Switzerland [4] and the Snowball distributed file system at the University of Kentucky [180].

2.2 Peer-to-Peer Systems

The popularity of Peer-to-Peer has led to a number of (often contradictory) definitions.

Tim O'Reilly in [139, p.40] says:

Ultimately, Peer-to-Peer is about overcoming the barriers to the formation of ad hoc communities, whether of people, of programs, of devices, or of distributed resources.

The definition [186] currently adopted by Wikipedia [187] says that:

A Peer-to-Peer (or P2P) computer network is a network that relies primarily on the computing power and bandwidth of the participants in the network rather than concentrating it in a relatively low number of servers.

Shirky in [161, p.22] defines Peer-to-Peer as (its definition has also been adopted by Ian Foster in [71]):

Peer-to-Peer is a class of applications that takes advantage of resources — storage, cycles, content, human presence — available at the edges of the Internet.

The Shirky's definition, as suggested by Androutsellis-Theotokis and Spinellis in [11], encompasses systems that completely rely upon centralized servers for their operation (such as SETI@home¹, various instant messaging systems, or even the notorious Napster), as well as various applications from the field of Grid computing.

Steinmetz refined the Shirky's definition in [163]:

¹<http://setiathome.berkeley.edu/>

[a Peer-to-Peer system is] *a self-organizing system of equal, autonomous entities (peers) [which] aim for the shared usage of distributed resources in a networked environment avoiding central services.*

Moreover, Steinmetz and Wehrle in [165] give an equivalent shorter definition:

[a Peer-to-Peer system is] *a system with completely decentralized self-organization and resource usage.*

The Steinmetz and Wehrle's definition is the one that we prefer and thus, it will be used in the rest of the thesis. While the other ones can be more expressive, the Steinmetz and Wehrle's definition is the most technical one.

In [11], the lack of agreement on a definition – or rather the acceptance of various different definitions -- is attributed to the fact that systems or applications are labeled Peer-to-Peer not because of their internal operation or architecture whether they give the impression of providing direct interaction between computers. As a result, different definitions of Peer-to-Peer are applied to accommodate the various different cases of such systems or applications. However, they too propose a definition:

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.

This definition is meant to encompass degrees of centralization ranging from the pure, completely decentralized systems such as Gnutella, to partially centralized systems such as Kazaa².

²<http://www.kazaa.com>

Peer-to-Peer gained visibility with Napster's support for music sharing on the Web and its lawsuit with the music companies. However, the Peer-to-Peer approach is by no means just a technology for file sharing. Rather,

it forms a fundamental design principle for distributed systems. It clearly reflects the paradigm shift from coordination to cooperation, from centralization to decentralization, and from control to incentives. Incentive-based systems raise a large number of important research issues. Finding a fair balance between give and take among peers may be crucial to the success of this technology. [164]

There are a number of books published about Peer-to-Peer systems [138, 123, 68, 54, 128, 17, 169, 100], mobile networks [140, 85], Peer-to-Peer Information Retrieval [see 7, 173]. Good surveys of Peer-to-Peer systems are [124, 11].

2.2.1 Characterization

As suggested by Steinmetz and Wehrle in [165], Peer-to-Peer systems are characterized as follows (though a single system rarely exhibits all of these properties):

Decentralized Resource Usage:

1. Resources of interest (bandwidth, storage, processing power) are used in a manner as equally distributed as possible and are located at the edges of the network, close to the peers. Thus, with regard to network topology, Peer-to-Peer systems follow the *end-to-end arguments* [155] which are the main reasons for the success of the Internet.
2. Within a set of peers, each utilizes the resources provided by other peers. The most prominent examples for such resources are storage and processing capacity. Other possible resources are connectivity, human presence, or geographic proximity

(with instant messaging and group communication as application examples).

3. Peers are interconnected through a network and in most cases distributed globally.
4. A peer's Internet address typically changes so the peer is not constantly reachable at the same address (transient connectivity). Often, they may be disconnected or shut down over longer periods of time. Among other reasons, this encourages Peer-to-Peer systems to introduce new address and name spaces above of the traditional Internet address level. Hence, content is usually addressed through unstructured identifiers derived from the content with a hash function. Consequently, data is no longer addressed by location (the address of the server) but by the data itself. With multiple copies of a data item, queries may locate any one of those copies. Thus Peer-to-Peer systems locate data based on content in contrast to location-based routing in the Internet.

Decentralized Self-Organization:

1. In order to utilize shared resources, peers interact directly with each other. In general, this interaction is achieved without any central control or coordination. Peer-to-Peer systems establish a cooperation between equal partners.
2. Peers directly access and exchange the shared resources they utilize without a centralized service. Thus Peer-to-Peer systems represent a fundamental decentralization of control mechanisms. However, performance considerations may lead to centralized elements being part of a complete Peer-to-Peer system, e.g. for efficiently locating resources. Such systems are commonly called server-based or centralized Peer-to-Peer systems (see Figure 2.1 [p.23]).
3. In a Peer-to-Peer system, peers can act both as clients and servers. This is radically different from traditional systems

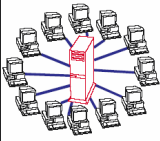
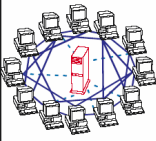
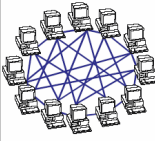
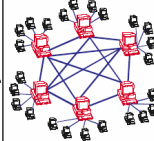
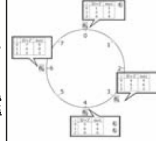
Client-Server	Peer-to-Peer			
	1. Resources are shared between the peers 2. Resources can be accessed directly from other peers 3. Peer is provider and requestor (Servent concept)			
	Unstructured P2P			Structured P2P
	1st Generation		2nd Generation	
1. Server is the central entity and only provider of service and content. → Network managed by the Server 2. Server as the higher performance system. 3. Clients as the lower performance system Example: WWW	<i>Centralized P2P</i>	<i>Pure P2P</i>	<i>Hybrid P2P</i>	<i>DHT-Based</i>
	1. All features of Peer-to-Peer included 2. Central entity is necessary to provide the service 3. Central entity is some kind of index/group database Example: Napster	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities Examples: Gnutella 0.4, Freenet	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → dynamic central entities Example: Gnutella 0.6, JXTA	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities 4. Connections in the overlay are "fixed" Examples: Chord, CAN
				

Figure 2.1: Summary of the characteristic features of Client-Server and Peer-to-Peer networks

with asymmetric functionality. It leads to additional flexibility with regard to available functionality and to new requirements for the design of Peer-to-Peer systems.

4. Peers are equal partners with symmetric functionality. Each peer is fully autonomous regarding its respective resources.
5. Ideally, resources can be located without any central entity or service. Similarly, the system is controlled in a self-organizing or ad hoc manner. As mentioned above, this guide line may be violated for reasons of performance. However, the decentralized nature should not be violated. The result of such a mix is a Peer-to-Peer system with a server-based approach.

As shown in Figure 2.1 (taken from [59]), in a Client-Server system the server is the only provider of service or content. The peers (clients) in this context only request content or service from the server. The clients do not provide any service or content to

run this system. Thus generally the clients are lower performance systems and the server is a high performance system. In contrast, in Peer-to-Peer systems all resources, i.e. the shared content and services, are provided by the peers. Some central facility may still exist, e.g. to locate a given content. A *peer* in this context is simply “*an application running on a machine*” [59], which may be a personal computer, a handheld or a mobile phone. In contrast to a Client-Server network, it is generally not possible to distinguish between a content requestor (client) and a content provider, as one application participating in the overlay in general offers content to other peers and requests content from other participants. This is often expressed by the term *servent*, composed of the first syllable of the term Server and the second syllable of the term Client.

Using this basic concept Figure 2.1 outlines various possibilities currently used. In the first generation centralized Peer-to-Peer systems some central server is still available. However, this server only stores the IP addresses of peers where some content is available. The address of that server must be known to the peers in advance. This concept was widely used and became especially well known due to Napster, offering free music downloads by providing the addresses of peers sharing the desired content. This approach subsequently lost much of its importance due to legal issues.

As a replacement for that scheme decentrally organized schemes, generally termed *pure Peer-to-Peer* such as Gnutella 0.4 and Freenet³ [52] became widely used. These schemes do not rely on any central facility (except possibly for some bootstrap server to ease joining such a network), but rely on flooding the desired content identifier over the network. Contacted peers that share that content will then respond to the requesting peer which will subsequently initiate a separate download session. These schemes generate a potentially huge amount of signaling traffic by flooding the requests.

To avoid that, schemes like Gnutella 0.6 or JXTA⁴ [76] introduced a hierarchy by defining *superpeers*, which store the content available at the connected peers together with their IP address. The

³<http://freenetproject.org/>

⁴<http://www.jxta.org/>

superpeers are often able to answer incoming requests by immediately providing the respective IP address, so that on average less hops are required in the search process, thus reducing the signaling traffic. This *second generation* Peer-to-Peer systems are generally named *hybrid*.

2.2.2 The lookup problem

Because of their completely decentralized character, the distributed coordination of resources is a major challenge in Peer-to-Peer systems. One of the research problems is: *How do you find any given data item x in a large Peer-to-Peer system in a scalable manner, without any centralized server or hierarchy?* [15]

More generally, x may be some (small) data item, the location of some bigger content, or coordination data, e.g. the current status of a node, or its current IP address, etc. This problem is the heart of any Peer-to-Peer system and is called the *lookup problem*.

More precisely the lookup problem can be defined as:

Given a data item x stored at some dynamic set of nodes in the system, find it. [15]

As underlined in [181], interesting questions about Peer-to-Peer systems and the lookup problem are:

- Where should a node store a given data item x ?
- How do other nodes discover the location of x ?
- How can the distributed system⁵ be organized to assure scalability and efficiency?

Four strategies have been proposed in literature to store and retrieve data in distributed systems: *server-based*, *hybrid Peer-to-Peer*, *flooding search* (also called *pure Peer-to-Peer*), and *distributed indexing* (also referred as *structured Peer-to-Peer* has been adopted by DHTs).

⁵ In the context of Peer-to-Peer systems, the distributed system – the collection of participating nodes pursuing the same purpose – is often called the *overlay network* or *overlay system*.

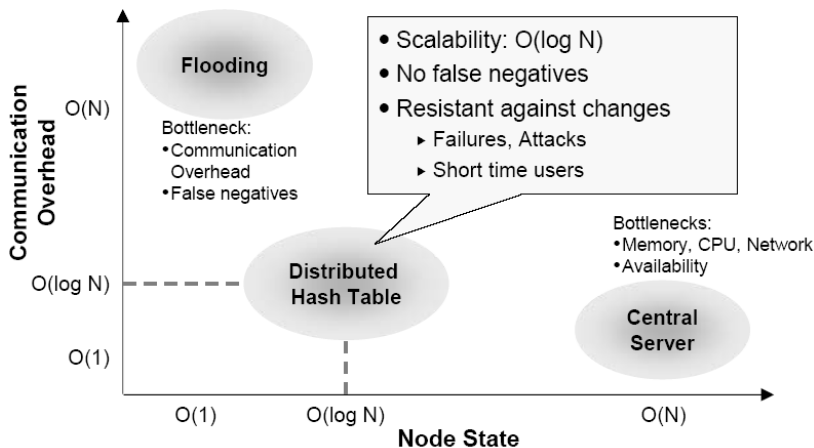


Figure 2.2: Strategies to store and retrieve data in distributed systems comparison.

In Figure 2.2 [181] the three basic strategies (the hybrid is missed) are compared considering the communication overhead and storage cost per node. Also bottlenecks and special characteristics of each approach are named.

Often, systems that adopt centralized and pure approaches are referred as *first generation Peer-to-Peer systems* while the flooding search is considered typical of *second generation Peer-to-Peer systems*. Both first and second generation Peer-to-Peer systems are generally termed *unstructured* because the content stored on a given node and its IP address are unrelated and do not follow any specific structure. Instead, systems that adopt the distributed indexing approach are referred as *structured Peer-to-Peer systems*. In Section 2.3 [p.27] and Section 2.4 [p.29] we analyze structured and unstructured Peer-to-Peer systems respectively.

2.3 Unstructured Peer-to-Peer Systems

Both first generation (i.e. *server-based* and *pure*) and second generation (i.e. *hybrid*) (see Figure 2.1 [p.23]) Peer-to-Peer systems are generally termed “*Unstructured*”, because the content stored on a given node and its IP address are unrelated and do not follow any specific structure.

The server-based approach was adopted by the first Peer-to-Peer-based file sharing applications. The data was transferred directly between peers, only after looking up the location of a data item via the server (Centralized P2P, cf. Figure 2.1 [59]).

First generation Peer-to-Peer systems, such as Napster, maintain the current locations of data items in a central server. After joining the Peer-to-Peer system, a participating node submits to the central server information about the content it stores and/or the services it offers. Thus requests are simply directed to the central server that responds to the requesting node with the current location of the data. The transmission of the located content is then organized in a Peer-to-Peer fashion between the requesting node and the node storing the requested data.

The centralized server approach has the advantage of retrieving the location of the desired information with a search complexity of $O(1)$. Also, fuzzy and complex queries are possible, since the server has a global overview of all available content. However, the central server is a critical element within the whole system concerning scalability and availability. The complexity in terms of memory consumption is $O(|\mathcal{X}|)$, with $|\mathcal{X}|$ representing the number of items available in the distributed system. The server also represents a single point of failure and attack.

Second generation of Peer-to-Peer systems (e.g. Gnutella⁶ 0.4 and Freenet⁷[52]) pursued an opposite approach. They keep no explicit information about the location of data items in other nodes. This means that there is no additional information concerning where to find a specific item in the distributed system. Thus, to retrieve

⁶<http://www.gnutella.com/>

⁷<http://freenetproject.org/>

a data item, the only chance is to ask as much participating nodes as necessary, whether or not they presently have the requested data. Thus, in Peer-to-Peer systems, a request is broadcasted among the nodes of the distributed system. If a node receives a query, it floods this message to other nodes until a certain hop count (Time to Live TTL) is exceeded.

Often, the general assumption is that content is replicated multiple times in the network, so a query may be answered in a small number of hops. A well-known example of such an application is Gnutella [1]. Gnutella includes several mechanisms to avoid request loops, but it is obvious that such a broadcast mechanism does not scale well. The number of messages and the bandwidth consumed is extremely high and increases more than linearly with increasing numbers of participants. In fact, after the central server of Napster was shut down in July 2001 due to a court decision [185], an enormous number of Napster users migrated to the Gnutella network within a few days, and under this heavy network load the system collapsed [15].

The advantage of flooding-based systems, such as Gnutella, is that there is no need for proactive efforts to maintain the network. Also, unsharp queries can be placed, and the nodes implicitly use proximity due to the expanding search mechanism. Furthermore, there are efforts to be made when nodes join or leave the network. But still the complexity of looking up and retrieving a data item is $O(|\mathcal{X}|^2)$, or even higher, and search results are not guaranteed, since the lifetime of request messages is restricted to a limited number of hops.

On the other hand, storage cost is in the order of $O(1)$ because data is only stored in the nodes actually providing the data – whereby multiple sources are possible – and no information for a faster retrieval of data items is kept in intermediate nodes. Overall, flooding search is an adequate technique for file-sharing-like purposes and complex queries.

It is apparent that neither approach scales well. The server-based system suffers from exhibiting a single point of attack as well as being a bottleneck with regard to resources such as memory, pro-

cessing power, and bandwidth, while the flooding-based approaches show tremendous bandwidth consumption on the network. Generally, these unstructured systems were developed in response to user demands (mainly file sharing and instant messaging) and consequently suffer from ad hoc designs and implementations.

2.4 Structured Peer-to-Peer Systems

Contrary to the *unstructured Peer-to-Peer* systems, also Peer-to-Peer approaches have been proposed which establish a link between the stored content and the IP address of a node. These systems are generally termed “structured Peer-to-Peer systems” (see Figure 2.1 [p.23]). The link between a content identifier and the IP address is usually based on Distributed Hash Tables (DHTs).

2.4.1 Introduction to DHTs

As Wehrle, Gtz, and Rieche said in [181],

a Distributed Hash Table manages data by distributing it across a number of nodes and implementing a routing scheme which allows efficient look up the node on which a specific data item is located. In contrast to flooding-based searches in unstructured systems, each node in a DHT becomes responsible for a particular range of data items. Also, each node stores a partial view of the whole distributed system which effectively distributes the routing information.

The DHT functionality, is already proving to be a useful substrate for large distributed systems. DHTs have already been used for distributed file systems [58, 151, 98] application-layer multicast [146, 195], event notification services [36, 42].

The routing procedure, using the partial view stored by each node, typically traverses several nodes, getting closer to the destination until the destination node is reached. Thus DHTs follow a

proactive strategy for data retrieval by structuring the search space and providing a deterministic routing scheme.

As reported in [181], overall, DHTs possess the following characteristics:

- In contrast to unstructured Peer-to-Peer systems, each DHT node manages a small number of references to other nodes. By means these are $O(\log(N))$ references, where N depicts the number of nodes in the system.
- By mapping nodes and data items into a common address space, routing to a node leads to the data items for which a certain node is responsible.
- Queries are routed via a small number of nodes to the target node. Because of the small set of references each node manages, a data item can be located by routing via $O(\log(N))$ hops. The initial node of a lookup request may be any node of the DHT.
- By distributing the identifiers of nodes and data items nearly equally throughout the system, the load for retrieving items should be balanced equally among all nodes.
- Because no node plays a distinct role within the system, the formation of hot-spots or bottlenecks can be avoided. Also, the departure or dedicated elimination of a node should have no considerable effects on the functionality of a DHT. Therefore, DHTs are considered to be very robust against random failures and attacks.
- A distributed index provides a definitive answer about results. If a data item is stored in the system, the DHT guarantees that the data is found.

Address space

DHTs introduce new address spaces into which data is mapped. Address spaces typically consist of large integer values. DHTs

achieve distributed indexing by assigning a contiguous portion of the address space to each participating node (Figure 7.6). Given a value from the address space, the main operation provided by a DHT system is the lookup function (see Subsection 2.2.2 [p.25]).

The mayor difference between DHT approaches is how they internally manage and partition their address space. In most cases, these schemes lend themselves to geometric interpretations of address spaces.

In a DHT system, each data item is assigned an identifier ID, a unique value from the address space. This value can be chosen freely by the application, but it is often derived from the data itself (e.g. the complete binary file or the file name) via a collision-resistant hash function, such as SHA-1 [137]. Thus the DHT would store the file at the node responsible for the portion of the address space which contains the identifier.

Based on the lookup function, most DHTs also implement a storage interface similar to a hash table. Thus the *put* function accepts an identifier and arbitrary data to store the data. This identifier and the data is often referred to as (key,value)-tuple. Symmetrically, the *get* function retrieves the data associated with a specified identifier.

DHTs can be used for a wide variety of applications. Applications are free to associate arbitrary semantics with identifiers, e.g. hashes of search keywords, database indexes, geographic coordinates, hierarchical directory like binary names, etc. Thus such diverse applications as distributed file systems, distributed databases, and routing systems have been developed on top of DHTs, e.g. application-layer multicast [95], email systems [125], storage systems [98, 58], indirection infrastructures [166], etc.

Load balancing

Most DHT systems attempt to spread the load of routing messages and of storing data on the participating nodes evenly [150, 135]. However, as suggested in [181], there are at least three reasons why some nodes in the system may experience higher loads than others:

- a node manages a very large portion of the address space,
- a node is responsible for a portion of the address space with a very large number of data items,
- or a node manages data items which are particularly popular.

Under these circumstances, additional load-balancing mechanisms can help to spread the load more evenly over all nodes. For example, a node may transfer responsibility for a part of its address space to other nodes, or several nodes may manage the same portion of address space. In [149] load-balancing schemes are discussed in details.

Routing

The fundamental principle of the large variety of approaches to routing implemented by the DHT system is to provide each node with a limited view of the whole system. Moreover, a bounded number of links to other nodes are stored on each node. When a node receives a message for a destination ID it is not responsible for itself, it forwards the message to one of these other nodes it stores information about. This process is repeated recursively until the destination node is found.

The routing algorithm and the routing metric determine the next-hop node. A typical metric is that of numeric closeness, i.e. messages are always forwarded to the node managing the identifiers numerically closest to the destination ID. A challenge for routing algorithms and metrics designing is that node failures and incorrect routing information have limited or little impact on routing correctness and system stability.

A key difference in the algorithms is the data structure that they use as a routing table to provide $O(\log(N))$ lookups.

Chord maintains a data structure that resembles a skiplist. Each node in Kademlia, Pastry, and Tapestry maintains a tree-like data structure. Viceroy maintains a butterfly data structure, which requires information about only constant other number nodes, while

still providing $O(\log(N))$ lookup. A recent variant of Chord (Kororde [91]), uses de Bruijn graphs, which requires each node to know only about two other nodes, while also providing $O(\log(N))$ lookup.

Data Storage

There are two possibilities for storing data in a DHT. In a DHT which uses *direct storage*, the data is copied upon insertion to the node responsible for it. In this case the node which inserted it can subsequently leave the DHT without the data becoming unavailable. However there is an overhead in terms of storage and network bandwidth. Since nodes may fail, the data must be replicated to several nodes to increase its availability.

The other possibility is to store references to the data. The inserting node only places a pointer to the data into the DHT. The data itself remains on this node. However, the data is only available as long as the node is available.

In both cases, the node using the DHT for lookup purposes does not have to be part of the DHT in order to use its services. Thus it is possible to realize a DHT service as third-party infrastructure service (see OpenDHT Project [148]).

Node Arrival

Generally speaking, it takes four steps for a node to join a DHT.

1. The new node has to get in contact with the DHT.
2. The new node gets to know some arbitrary node of the DHT using a bootstrap method (this node is used as an entry point to the DHT until the new node is an equivalent member of the DHT).
3. A partition in the logical address space is assigned to the new node (the routing information in the system needs to be updated to reflect the presence of the new node).
4. The new node retrieves all (key, value) pairs under its responsibility from the node that stored them previously.

Depending on the DHT implementation, a node may choose arbitrary or specific partitions on its own or not.

Node Failure

Because DHTs are often composed of poorly connected desktop machines, failures are usually assumed to occur frequently. Thus all non-local operations in a DHT need to resist failures of other nodes, reflecting the self-organizing design of DHT algorithms. For example, routing and lookup procedures are typically designed to use alternative routes toward the destination when a failed node is encountered on the default route. Many DHTs also employ proactive recovery mechanisms, e.g. to maintain their routing information. Consequently, they periodically probe other nodes to check whether these nodes are still operational. Furthermore, node failures lead to a re-partitioning of the DHT's address space. This may in turn require (key, value)-pairs to be moved between nodes and additional maintenance operations such as adaptation to new load-balancing requirements. When a node fails, the application data that it stored is lost unless the DHT uses replication to keep multiple copies on different nodes. Some DHTs follow the simpler soft-state approach which does not guarantee persistence of data. Data items are pruned from the DHT unless the application refreshes them periodically. Therefore, a node failure leads to a temporary loss of application data until the data is refreshed.

Node Departure

Nodes which voluntarily leave a DHT could be treated the same as failed nodes. However, DHT implementations often require departing nodes to notify the system before leaving. This allows other nodes to copy application data from the leaving node and to immediately update their routing information leading to improved routing efficiency.

Performance measures

In [91] five performance measures for DHTs are considered:

- *Degree*: the number of neighbors with which a node must maintain continuous contact;
- *Hop Count*: the number of hops needed to get a message from any source to any destination;
- The degree of *Fault Tolerance*: what fraction of the nodes can fail without eliminating data or preventing successful routing;
- The *Maintenance Overhead*: how often messages are passed between nodes and neighbors to maintain coherence as nodes join and depart;
- The degree of *Load Balance*: how evenly keys are distributed among the nodes, and how much load each node experiences as an intermediate node for other routes.

It is important to note that optimizing one tends to put pressure on the others. In other words, we agree with [181] that the design challenges are:

- *Routing efficiency*: The latency of routing and lookup operations is influenced by the topology of the address space, the routing algorithm, the number of references to other nodes, the awareness of the IP-level topology, etc.
- *Management overhead*: The costs of maintaining the Distributed Hash Table under no load depend on such factors as the number of entries in routing tables, the number of links to other nodes, and the protocols for detecting failures.
- *Dynamics*: A large number of nodes joining and leaving a Distributed Hash Table often referred to as churn concurrently puts particular stress on the overall stability of the system, reducing routing efficiency, incurring additional management traffic, or even resulting in partitioned or defective systems.

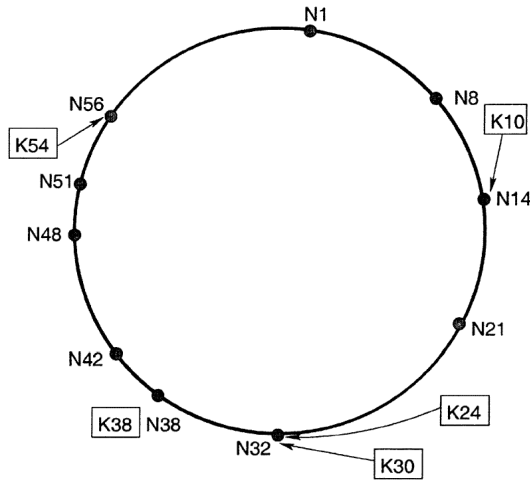


Figure 2.3: Chord identifier circle (ring) consisting of ten nodes storing five keys.

2.4.2 Chord

Chord, which is probably the most famous DHTs, has been defined in [167] and [168]. The elegance of the Chord algorithm derives from its simplicity. In Chord, the keys are l -bit identifiers (IDs) that form a one-dimensional circle. Both data items and nodes are associated with an ID. The (key, value) pair (k, v) is hosted by the node whose ID is greater than or equal to key k . Such a node is called the successor of key k . In other words, a node in a Chord circle with clockwise increasing IDs is responsible for all keys that precede it counter-clockwise.

In Chord each node stores its successor node on the identifier circle. When a key is being looked up, each node forwards the query to its successor in the identifier circle until one of the nodes determines that the key lies between itself and its successor. The successor is communicated as the result of the query back to its originator (the key must be hosted by this successor).

To achieve scalable key lookup, each node also maintains information about no-successor nodes in a routing table called finger

table. Given a circle with l -bit identifiers, a finger table has a maximum of l entries. Thus its size is independent of the number of keys or nodes forming the DHT. Each finger entry consists of a node ID, an IP address and port pair, and possibly some book-keeping information. The routing information from finger tables provides information about nearby nodes and a coarse-grained view of long-distance links at intervals increasing by powers of two. Using the finger tables queries are routed over large distances on the identifier circle in a single hop. In fact a node forwards queries for a given key to the closest predecessor according to its finger table.

Furthermore, the closer the query gets to k , the more accurate the routing information of the intermediate nodes on the location of k becomes. Given the power-of-two intervals of finger IDs, each hop covers at least half of the remaining distance on the identifier circle between the current node and the target identifier. This results in an average of $O(\log(N))$ routing hops for a Chord circle with N participating nodes. Stoica et al. show that the average lookup requires $1/2 \log(N)$ steps.

In order to join a Chord identifier circle, the new node first determines some identifier \underline{n} . The original Chord protocol does not impose any restrictions on this choice. There have been several proposals to restrict node IDs according to certain criteria, e.g. to exploit network locality or to avoid identity spoofing. For the new node \underline{n} , another node \underline{n}_0 must be known which already participates in the Chord system. By querying \underline{n}_0 for \underline{n} 's own ID, \underline{n} retrieves its successor. It notifies its successor \underline{n}_s of its presence leading to an update of the predecessor pointer of \underline{n}_s to \underline{n} . Node \underline{n} then builds its finger by iteratively querying o for the successors of $\underline{n} + 21$, $\underline{n} + 22$, $\underline{n} + 23$, etc. At this stage, \underline{n} has a valid successor pointer and finger table. However, \underline{n} does not show up in the routing information of other nodes. In particular, it is not known to its predecessor as its new successor since the lookup algorithm is not apt to determine a node's predecessor.

Chord introduces a stabilization protocol to validate and update successor pointers as nodes join and leave the system. Stabilization requires an additional predecessor pointer and is performed peri-

odically on every node. With the stabilization protocol, the new node \underline{n} does not actively determine its predecessor. Instead, the predecessor itself has to detect and fix inconsistencies of successor and predecessor pointers. After the new node has thus learnt of its predecessor, it copies all keys it is responsible for. At this stage, all successor pointers are up to date and queries can be routed correctly, albeit slowly. Since the new node \underline{n} is not present in the finger tables of other nodes, they forward queries to the predecessor of \underline{n} even if \underline{n} would be more suitable. Node \underline{n} 's predecessor then needs to forward the query to \underline{n} via its successor pointer. Chord updates finger tables lazily. Each node periodically picks a finger randomly from the finger table at index $i(1 < i = l)$ and looks it up to find the true current successor of $\underline{n} + 2i - 1$.

Chord addresses node failures on several levels. When a node detects a failure of a finger during a lookup, it chooses the next best preceding node from its finger table. Failed nodes are removed from the finger tables. It is particularly important to maintain the accuracy of the successor information as the correctness of lookups depends on it. To maintain a valid successor pointer in the presence of multiple simultaneous node failures, each node holds a successor list of a certain length that contains the node's first successors. When a node detects the failure of its successor, it reverts to the next live node in its successor list. The Chord ring is affected only if all nodes from a successor list fail simultaneously. The successor of a failed node becomes responsible for the keys and data of the failed node. Thus an application utilizing Chord can replicate data to successor nodes to reduce the risk of data lost. Chord can use the successor list to communicate this information and possible changes to the application.

In Chord a leaving node transfer its keys to its successor and notify its successor and predecessor. This ensures that data is not lost and that the routing information remains intact.

Important variations of Chord are Symphony Subsection 2.4.9 [p.46] and Viceroy Subsection 2.4.10 [p.47]. A recent variant of Chord using the De Bruijn graphs is Koorde [91], which requires each node to know only about two other nodes, while also providing

0	1	2	3	4	5		7	8	9	a	b	c	d	e	f
x	x	x	x	x	x		x	x	x	x	x	x	x	x	x
<hr/>															
6	6	6	6	6		6	6	6	6	6	6	6	6	6	6
0	1	2	3	4		6	7	8	9	a	b	c	d	e	f
x	x	x	x	x		x	x	x	x	x	x	x	x	x	x
<hr/>															
6	6	6	6	6	6	6	6	6	6		6	6	6	6	6
5	5	5	5	5	5	5	5	5	5		5	5	5	5	5
0	1	2	3	4	5	6	7	8	9		b	c	d	e	f
x	x	x	x	x	x	x	x	x	x		x	x	x	x	x
<hr/>															
6		6	6	6	6	6	6	6	6	6	6	6	6	6	6
5		5	5	5	5	5	5	5	5	5	5	5	5	5	5
a		a	a	a	a	a	a	a	a	a	a	a	a	a	a
0		2	3	4	5	6	7	8	9	a	b	c	d	e	f
x		x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figure 2.4: Routing table of a Pastry node with nodeID 65a1x, $b=4$. Digits in base 16, x represents an arbitrary suffix. The IP address associated with each entry is not shown.

$O(\log(N))$ lookup.

In [72], Chord has been extended to solve multi-dimensional attributed range queries transforming the original vector space into a single-dimensional domain. The method is called Space-filling Curves with Range Partitioning (SCRAP) and uses space filling curves in order to mapping.

2.4.3 Pastry

The Pastry distributed routing system was proposed by Rowstron and Druschel in [152]. In Pastry the routing is based on numeric closeness of identifiers. In their work, the authors focus not only on the number of routing hops, but also on network locality as factors in routing efficiency. In Pastry, nodes and data items uniquely associate with l -bit identifiers (l is typically 128). Pastry views identifiers as strings of digits to the base $2b$ where b is typically chosen to be 4. A key is located on the node to whose node ID it

is numerically closest.

Pastry's node state is divided into three main elements. The routing table, similar to Chord's finger table, stores links into the identifier space. The leaf set contains nodes which are close in the identifier space (like Chord's successor list). Nodes that are close together in terms of network locality are listed in the neighborhood set.

Pastry measures network locality based on a given scalar network proximity metric which is assumed to be already available from the network infrastructure. This metric might range from IP hops to actual the geographical location of nodes.

A Pastry node's routing table is made up of l/b rows with $2^b - 1$ entries per row. On node n , the entries in row i hold the identities of Pastry nodes whose node IDs share an i -digit prefix with n but differ in digit n itself. When there is no node with an appropriate prefix, the corresponding table entry is left empty.

As in Chord, in Pastry a node has a coarse-grained knowledge of other nodes which are distant in the identifier space. The detail of the routing information increases with the proximity of other nodes in the identifier space. Without a large number of nearby nodes, the last rows of the routing table are only sparsely populated. Intuitively, the identifier space would need to be fully exhausted with node IDs for complete routing tables on all nodes. In a system with N nodes, only $\log_{2^b}(N)$ routing table rows are populated on average.

In populating the routing table, there is a choice from the set of nodes with the appropriate identifier prefix. During the routing process, network locality can be exploited by selecting nodes which are close in terms of a network proximity metric.

To increase lookup efficiency, the leaf set L of node n holds the $|L|$ nodes numerically closest to n . The routing table and the leaf set are the two sources of information relevant for routing. The leaf set also plays a role similar to Chord's successor lists in recovering from failures of adjacent nodes.

Instead of numeric closeness, the neighborhood set M is concerned with nodes that are close to the current node with regard to

the network proximity metric. Thus it is not involved in routing itself but in maintaining network locality in the routing information.

Routing in Pastry is divided into two main steps:

1. a node checks whether the key k is within the range of its leaf set. If this is the case, it implies that k is located on one of the nearby nodes of the leaf set.
2. the node forwards the query to the leaf set node numerically closest to k . In case this is the node itself, the routing process is finished.

If k does not fall into the range of leaf set nodes, the query needs to be forwarded over a longer distance using the routing table. In this case, a node n tries to pass the query on to a node which shares a longer common prefix with k than n itself. If there is no such entry in the routing table, the query is forwarded to a node which shares a prefix with k of the same length as n but which is numerically closer to k than n .

Pastry optimizes two aspects of routing and locating the node responsible for a given key: it attempts both to achieve a small number of hops to reach the destination node, and to exploit network locality to reduce the overhead of each individual hop.

For detailed description of Pastry see [152, 37, 39, 41, 38, 114, 79] An open-source implementation of Pastry is FreePastry [175]. A scalable application-level multicast application called Scribe [42] has been built upon Pastry. In [125] ePost, a Peer-to-Peer email system built using Pastry, is presented.

2.4.4 Tapestry

Tapestry, defined by Zhao, Kubiawicz, and Joseph in [193], is very similar to Pastry (see Subsection 2.4.3 [p.39]) but differs in its approach to mapping keys to nodes in the sparsely populated id space, and in how it manages replication. In Tapestry, there is no leaf set and neighboring nodes in the namespace are not aware of each other. When a node's routing table does not have an entry for

a node that matches a key's n th digit, the message is forwarded to the node in the routing table with the next higher value in the n th digit modulo $2b$. This procedure, called *surrogate routing*, maps keys to a unique live node if the node routing tables are consistent. For fault tolerance, Tapestry inserts replicas of data items using different keys. The expected number of routing hops is $\log_6(N)$.

Tapestry has been also described in [194]. OceanStore, a utility infrastructure designed to span the globe and provide continuous access to persistent information, presented in [98] uses Tapestry.

In [195]), application-level multicast for Pastry called Bayeux is presented .

2.4.5 Chimera

Chimera [13] is a light-weight C implementation of a structured overlay that provides similar functionality as prefix-routing protocols Tapestry (see Subsection 2.4.4 [p.41]) and Pastry (see Subsection 2.4.3 [p.39]). Chimera gains simplicity and robustness from its use of Pastry's leafsets, and efficient routing from Tapestry's locality algorithms. In addition to these properties, Chimera also provides efficient detection of node and network failures, and reroutes messages around them to maintain connectivity and throughput.

2.4.6 Z-Ring

Z-Ring [104] is a fast prefix routing protocol for Peer-to-Peer overlay networks. The main idea in Z-Ring is to classify peers into different closed groups at each routing level. Z-Ring uses efficient membership maintenance to support one or two-hop key-based routing in large dynamic networks. The analysis and simulations presented in [104] show that it provides efficient routing with very low maintenance overhead.

Most of the DHTs show logarithmic number of hops which means that they scale well. However, with network size, the latency they incur can be substantial in practice. Z-Ring tries to achieve both goals of low routing hops and low maintenance costs in an

adaptive system by integrating Peer-to-Peer routing with efficient membership maintenance algorithms.

On each peer Z-Ring uses a membership protocol to maintain a large routing table, which is the set of active peers belonging to the same group. They started with Pastry (see Subsection 2.4.3 [p.39]) and expand its prefix routing base b from 16 to 4096. Using $b = 4096$, they can achieve one-hop routing with 4096 nodes and two-hop routing across 16 million peers. While traditional protocols require a node to periodically probe its links to each of its neighbors and thus make the maintenance of routing entries with $b = 4096$ infeasible, we leverage cost-efficient membership protocols to maintain routing entries and detect link or node failures.

2.4.7 Content Addressable Network *CAN*

We analyze in details the *CAN*, over which we built our *MCAN*, in Chapter 3 [p.63].

2.4.8 Kademlia

Presented in [118], Kademlia is probably the most used DHT. The following are known implementations of Kademlia (see [184] for Up-to-Date information):

- Overnet network used in *MLDonkey* and *eDonkey2000* (which is no longer available);
- Kad Network used by *eMule*⁸ (from v0.40), *MLDonkey*⁹ (from v2.5-28) and *aMule* (from v2.1.0);
- *RevConnect*¹⁰ (from v0.403);
- *KadC*¹¹. A C library to publish and retrieve information to and from the Overnet network;

⁸<http://www.emule-project.net/>

⁹<http://mldonkey.org/>

¹⁰<http://www.revconnect.com/>

¹¹<http://kadc.sourceforge.net/>

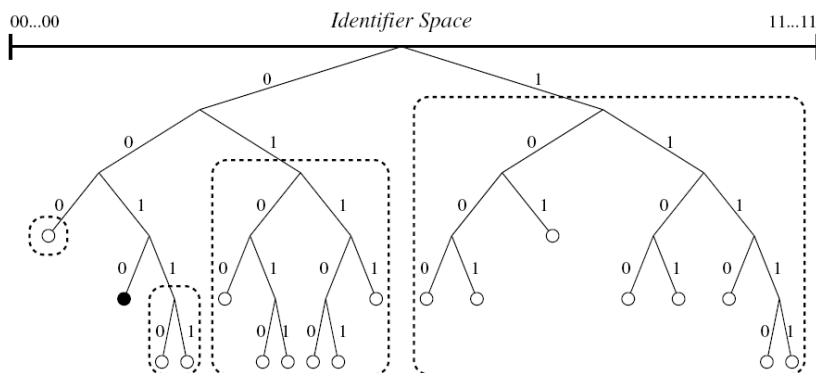


Figure 2.5: An example of a Kademlia topology

- Khashmir. A Python implementation of Kademlia.
- BitTorrent Azureus¹² DHT, a modified Kademlia implementation for decentralized tracking and various other fetures like the “Comments and Ratings” plugin used by Azureus (from v2.3.0.0);
- BitTorrent Mainline DHT, used by BitTorrent¹³ client (from v4.1.0), μ Torrent¹⁴ (from v1.2), BitSpirit¹⁵ (from v3.0) BitComet¹⁶ (from v0.59) and KTorrent¹⁷. They all share a DHT based on an implementation of the Kademlia algorithm, for trackerless torrents.

Many of Kademlia’s benefits result from its use of a XOR metric for distance between points in the key space. XOR is symmetric, allowing Kademlia participants to receive lookup queries from precisely the same distribution of nodes contained in their routing tables. Without this property, systems such Chord Subsection 2.4.2

¹²<http://sourceforge.net/projects/azureus/>

¹³<http://www.bittorrent.com/>

¹⁴<http://www.utorrent.com/>

¹⁵<http://www.167bt.com/intl/>

¹⁶<http://www.bitcomet.com>

¹⁷<http://ktorrent.org/>

[p.36] do not learn useful routing information from queries they receive. Worse yet, asymmetry leads to rigid routing tables. Each entry in a Chord node's finger table must store the precise node preceding some interval in the ID space. Any node actually in the interval would be too far from nodes preceding it in the same interval. Kademia, in contrast, can send a query to any node within an interval, allowing it to select routes based on latency or even send parallel, asynchronous queries to several equally appropriate nodes.

In their work on Kademia [118], Maymounkov and Mazières observe a mismatch in the design of Pastry: its routing metric (identifier prefix length) does not necessarily correspond to the actual numeric closeness of identifiers. As a result, Pastry requires two routing phases which impacts routing performance and complicates formal analysis. Thus Kademia uses an XOR routing metric which improves on these problems and optionally offers additional parallelism for lookup operations.

Kademia most resembles Pastry's (see Subsection 2.4.3 [p.39]) first phase, which successively finds nodes roughly half as far from the target ID by Kademia's XOR metric. In a second phase, however, Pastry switches distance metrics to the numeric difference between IDs. It also uses the second, numeric difference metric in replication. Unfortunately, nodes close by the second metric can be quite far by the first, creating discontinuities at particular node ID values, reducing performance, and complicating attempts at formal analysis of worst-case behavior.

Kademia's XOR metric measures the distance between two IDs by interpreting the result of the bit-wise exclusive OR function on the two IDs as integers. For example, the distance between the identifiers 3 and 5 is 6. Considering the shortest unique prefix of a node identifier, this metric effectively treats nodes and their identifiers as the leaves of a binary tree. For each node, Kademia further divides the tree into subtrees not containing the node, as illustrated in Figure 2.5 [182].

Each node knows of at least one node in each of the subtrees. A query for an identifier is forwarded to the subtree with the longest matching prefix until the destination node is reached. Similar to

Chord, this halves the remaining identifier space to search in each step and implies a routing latency of $O(\log(N))$ routing hops on average. In many cases, a node knows of more than a single node per subtree. Similar to Pastry, the Kademlia protocols suggests forwarding queries to α nodes per subtree in parallel. By biasing the choice of nodes towards short round-trip times, the latency of the individual hops can be reduced. With this scheme, a failed node does not delay the lookup operation. However, bandwidth usage is increased compared to linear lookups.

When choosing remote nodes in other subtrees, Kademlia favors old links over nodes that only recently joined the network. This design choice is based on the observation that nodes with long uptime have a higher probability of remaining available than fresh nodes. This increases the stability of the routing topology and also prevents good links from being flushed from the routing tables by distributed denial-of-service attacks, as can be the case in other DHT systems.

With its XOR metric, Kademlia's routing has been formally proved consistent and achieves a lookup latency of $O(\log(N))$. The required amount of node state grows with the size of a Kademlia network. However, it is configurable and together with the adjustable parallelism factor allows for a trade-off of node state, bandwidth consumption, and lookup latency.

2.4.9 Symphony

The Symphony protocol, defined in [116], is a variation of Chord Subsection 2.4.2 [p.36] that exploits the small world phenomenon. In Symphony each node establishes only a constant number of links to other nodes. This basic property of Symphony significantly reduces the amount of per-node state and network traffic when the overlay topology changes. However, with an increasing number of nodes, it does not scale as well as Chord.

In Symphony, the Chord finger table is replaced by a constant but configurable number k of long distance links which are chosen randomly according to harmonic distributions (hence the

name Symphony). Effectively, the harmonic distribution of long-distance links favors large distances in the identifier space for a system with few nodes and decreasingly smaller distances as the system grows. In Symphony a query is forwarded to the node with the shortest distance to the destination key.

Symphony additionally employs a 1-lookahead approach. The lookahead table of each node records those nodes which are reachable through the successor, predecessor, and long distance links, i.e. the neighbors of a node's neighbors. Instead of routing greedily, a node forwards messages to its direct neighbor (not a neighbor's neighbor) which promises the best progression towards the destination. This reduces the average number of routing hops by 40% at the expense of management overhead when nodes join or leave the system. The main contribution is its constant degree topology resulting in very low costs of per-node state and of node arrivals and departures. It also utilizes bidirectional links between nodes and bi-directional routing. Symphony's routing performance $O(1/k \log_2(N))$. However, nodes can vary the number of links they maintain to the rest of the system during run-time based on their capabilities, which is not permitted by the original designs of Chord, Pastry, and CAN.

2.4.10 Viceroy

In 2002, Malkhi, Naor, and Ratajczak proposed Viceroy [115], another variation on Chord (see Subsection 2.4.2 [p.36]). Viceroy maintains a butterfly data structure, which requires information about only constant other number nodes. Viceroy improves on the original Chord algorithm through a hierarchical structure of the ID space with constant degree which approximates a butterfly topology. This results in less per-node state and less management traffic but slightly lower routing performance than Chord. Viceroy borrows from Chord's fundamental ring topology with successor and predecessor links on each node. It also introduces a new node state called a level.

The routing procedure is split into three phases closely related

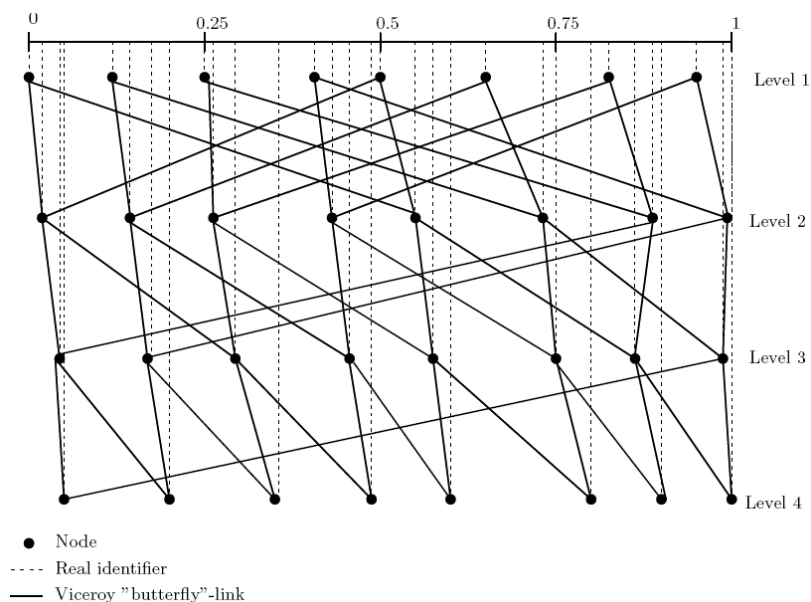


Figure 2.6: An ideal Viceroy network. Up and ring links are omitted for simplicity.

to the available routing information. First, a query is forwarded to level one along the uplinks. Second, a query recursively traverses the downlinks towards the destination. On each level, it chooses the downlink which leads to a node closer to the destination, without overshooting it in the clockwise direction. After reaching a node without downlinks, the query is forwarded along ringlevel and successor links until it reaches the target identifier.

The authors of Viceroy show that this routing algorithm yields an average number of $O(\log(N))$ routing hops. Like Symphony (see Subsection 2.4.9 [p.46]), Viceroy features a constant degree linkage in its node state. However, every node establishes seven links whereas Symphony keeps this number configurable even at run-time. Furthermore and similar to Chord, the rigid layout of the identifier space requires more link updates than Symphony when nodes join or leave the system. At the same time, the scalability of

System	Routing Hops	Node State	Arrival	Departure
Chord	$O(\frac{1}{2}\log_2(N))$	$O(2\log_2(N))$	$O(\log_2^2(N))$	$O(\log_2^2(N))$
Pastry	$O(\frac{1}{b}\log_2(N))$	$O(\frac{1}{b}(2^b - 1)\log_2(N))$	$O(\log_{2^b}(N))$	$O(\log_b(N))$
CAN	$O(\frac{D}{2}N^{\frac{1}{b}})$	$O(2D)$	$O(\frac{D}{2}N^{\frac{1}{b}})$	$O(2D)$
Symphony	$O(\frac{c}{k}\log^2(N))$	$O(2k + 2)$	$O(\log^2(N))$	$O(\log^2(N))$
Viceroy	$O(\frac{c}{k}\log^2(N))$	$O(2k + 2)$	$O(\log_2(N))$	$O(\log_2(N))$
Kademlia	$O(\log_b(N))$	$O(b \cdot \log_b(N))$	$O(\log_b(N))$	$O(\log_b(N))$

Figure 2.7: Performance comparison of DHT systems. The columns show the averages for the number of routing hops during a key lookup, the amount of per-node state, and the number of messages when nodes join or leave the system.

its routing latency of $O(\log(N))$ surpasses that of Symphony, while not approaching that of Chord, Pastry, and CAN.

2.4.11 DHTs Comparison

In Figure 2.7 we report the performance comparison results of the most important DHTs. These results were presented by Wehrle, Gtz, and Rieche in [182]. Another comparison of DHTs can be found in [103].

2.4.12 DHTs Related Works

A great number of papers about particular aspects of DHTs can be found, e.g. *topology-aware routing* in [40], *scalable application-level multicast* in [43], *complex queries* in DHTs in [81]. *load balancing* in Peer-to-Peer systems [74] and [171].

An important project using Chord is MINERVA. In MINERVA¹⁸ each peer is considered autonomous and has its own local search engine with a crawler and a corresponding local index. Peers share

¹⁸<http://www.mpi-inf.mpg.de/departments/d5/software/minerva/>

their local indexes (or specific fragments of local indexes) by posting meta-information into the Peer-to-Peer network. This meta-information contains compact statistics and quality-of-service information, and effectively forms a global directory. However, this directory is implemented in a completely decentralized and largely self-organizing manner. More specifically, they are maintained as a distributed hash table (DHT) using Chord. The MINERVA peer engine uses the global directory to identify candidate peers that are most likely to provide good query results. A query posed by a user is forwarded to other peers for better result quality. The local results obtained from there are merged by the query initiator. The MINERVA Project has been presented in [26], while related papers are [27, 24, 23, 25, 28, 47, 141, 120, 119].

In [170] the Threshold Algorithm [60, 63] was applied to the query processing problem in a Peer-to-Peer environment in which inverted lists are mapped to a set of peers, by employing an underlying DHT substrate. In [192] the work was extended defining 5 different policies for distributed query evaluation: Simple Algorithm, Distributed Threshold Algorithm, Bloom Circle Threshold Algorithm, Bloom Petal Threshold Algorithm (BPTA) and Simple Bloom Petal Algorithm (SBPA). From the experiments they conducted, the BTPA approach appears to perform best for queries with up to 4 terms while SBPA is better for longer queries.

In [174] an information retrieval system called pSearch has been defined. It uses a technique called latent semantic indexing to reduce the dimensionality of the space and a singular value decomposition to transform and truncate the matrix of term vectors computed in the previous step. The obtained lower-dimensional vector space is then distributed using CAN (see Chapter 3 [p.63]).

2.4.13 P-Grid

While unstructured Peer-to-Peer systems have generated substantial interest because of their self-organization structured overlay networks like DHTs typically requires a higher degree of coordination among the nodes while constructing and maintaining the

overlay network. However, self-organizing process can also be used in the context of structured overlay networks. With such an approach structural properties are not guaranteed through localized operation, but emerge as a global property from a self-organization process.

P-Grid¹⁹, first presented by Aberer in [2], adopt this approach. P-Grid uses self-organizing processes for the initial network construction to achieve load-balancing properties as well as for maintenance to retain structural properties of the overlay network intact during changes in the physical network. In P-Grid the key space is recursively bisected to achieve balanced workload of partitions. Bisecting the key space induces a canonical tree structure which is used as the basis for implementing a standard, distributed. Therefore P-Grid is a tree-based Peer-to-Peer network. *prefix routing scheme* for efficient search.

The P-Grid authors claim that it differs from other DHT approaches in terms of practical applicability (especially in respect to dynamic network environments), algorithmic foundations (randomized algorithms with probabilistic guarantees), robustness, and flexibility. In [55] the problem of executing range queries over a structured overlay network on top of a tree abstraction is discussed. The approach is evaluated using P-Grid. Other P-Grid related papers are [2, 8, 4, 5, 6].

2.4.14 Small-World and Scale-Free

Small-world network is a generalization of the *small world phenomenon* to non-social networks. Formally, it is a class of where most nodes are not neighbors of one another, but most nodes can be reached from every other by a small number of hops or steps. In [94] a family of *Small-World Access Methods (SWAM)* have been proposed for efficient execution of various similarity-search queries, namely exact match, Range and Nearest Neighbor, on vector space with L_p metrics (see Subsection 1.3.1 [p.9]). Furthermore, the authors also describe the SWAM-V structure that partitions the data

¹⁹P-Grid Project: <http://www.p-grid.org/>

space in a Voronoi-like manner of neighboring cells.

2.4.15 Other Works

In [121] KLEE a novel algorithmic framework for distributed top-k queries has been presented. It addresses the efficient processing of top-k queries in wide-area distributed data repositories where the index lists for the attribute values (or text terms) of a query are distributed across a number of data peers. In KLEE each data item has associated with it a set of descriptors, text terms or attribute values, and there is a precomputed score for each pair of data item and descriptor. The inverted index list for one descriptor is the list of data items in which the descriptor appears sorted in descending order of scores. These index lists are the distribution granularity of the distributed system. Each index list is assigned to one peer (or, if we wish to replicate it, to multiple peers).

P2PR-tree (Peer-to-Peer R-tree), defined in [127], is a spatial index specifically designed for Peer-to-Peer systems. It is hierarchical and performs efficient pruning of the search space by maintaining minimal amount of information concerning peers that are far away and storing more information concerning nearby peers.

2.5 Metric Peer-to-Peer Structures

Very recently, four scalable distributed have been proposed similarity search structures for metric data. The first two structures adopt the basic *ball* and *Generalized Hyperplane* partitioning principles [176] and they are called the VPT* and the GHT*, respectively (see Subsection 2.5.1 [p.53]).

The other two apply transformation strategies — the metric similarity search problem is transformed into a series of range queries executed on existing distributed keyword structures, namely the CAN (described in Chapter 3 [p.63]) and the Chord (described in Subsection 2.4.2 [p.36]). By analogy, they have been called the MCAN, which is the object of this thesis, and the M-Chord (see Subsection 2.5.2 [p.57]). Each of the structures is able to execute

similarity queries for any metric dataset, and they all exploit parallelism for query execution.

Result of numerous experiments conducted on implementations of the VPT*, GHT*, MCAN and M-Chord systems over the same infrastructure of peer computers, have been reported in [22]. In this thesis they are reported on Chapter 6 [p.117].

2.5.1 GHT* and VPT*

In this section, we describe two distributed metric index structures — the GHT* [18, 20, 21, 19] and its extension called the VPT* [22]. Both of them exploit native metric partitioning principles using them to build a distributed binary tree [97].

In both the GHT* and the VPT*, the dataset is distributed among peers participating in the network. Every peer holds sets of objects in its storage areas called *buckets*. A bucket is a limited space dedicated to storing objects, e.g. a memory segment or a block on a disk. The number of buckets managed by a peer depends on its own potential.

Since both structures are dynamic and new objects can be inserted at any time, a bucket on a peer may reach its capacity limit. In this situation, a new bucket is created and some objects from the full bucket are moved to it. This new bucket may be located on a peer different from the original one. Thus the structures grow as new data come in.

The core of the algorithm lays down a mechanism for locating respective peers which hold requested objects. The component of the structure responsible for this navigation is called the *Address Search Tree* (AST), an instance of which is present at every peer. Whenever a peer wants to access or modify the data in the GHT* structure, it must first consult its own AST to get locations, i.e. peers, where the data resides. Then, it contacts the peers via network communication to actually process the operation.

Since we are in a distributed environment, it is practically impossible to maintain a precise address for every object in every peer. Thus the ASTs at the peers contain only limited navigation infor-

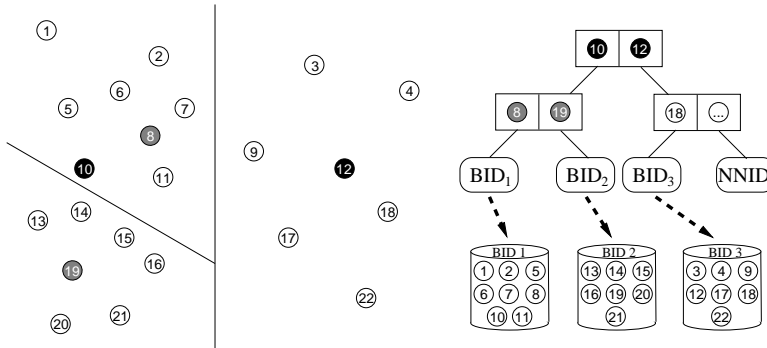


Figure 2.8: Address Search Tree with the generalized hyperplane partitioning

mation which may be imprecise. The locating step is repeated on contacted peers whenever AST is imprecise until the desired peers are reached. The algorithm guarantees that the destination peers are always found. Both of these structures also provide a mechanism called *image adjustment* for updating the imprecise parts of the AST automatically. For more technical details see [21].

Address Search Tree

The AST is a binary search tree based on the Generalized Hyperplane Tree (GHT) [176] in GHT*, and on the Vantage Point Tree (VPT) [176] for the VPT* structure. Its inner nodes hold the routing information according to the partitioning principle and each leaf node represents a pointer to either a local bucket (denoted as BID) or a remote peer (denoted as NNID) where the data of the respective partition is located.

An example of AST using the generalized hyperplane partitioning is depicted in Figure 2.8. In order to divide a set of objects $I = \{x_1, \dots, x_{22}\}$ into two separated partitions I_1, I_2 using the generalized hyperplane, we must first select a pair of objects from the set. In Figure 2.8, we select objects x_{10}, x_{12} and promote them to *pivots* of the first level of the AST. Then, the original set I is split

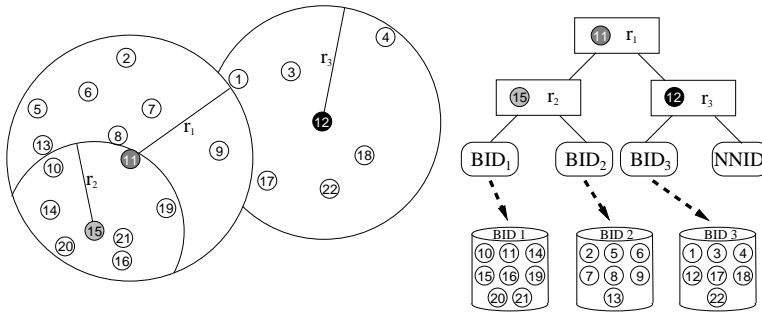


Figure 2.9: Address Search Tree with the vantage point partitioning

by measuring the distance between every object $x \in I$ and both the pivots. If $d(x, x_{10}) \leq d(x, x_{12})$, i.e. the object x is closer to the pivot x_{10} , the object is assigned to the partition I_1 and vice versa. This principle is used recursively until all the partitions are small enough and a binary tree representing the partitioning is built accordingly. Figure 2.8 shows an example of such a tree. Observe that the leaf nodes are denoted by BID_i and $NNID_i$ symbols. This means that the corresponding partition (which is small enough to stop the recursion) is stored either in a local bucket or on a remote peer respectively.

The vantage point partitioning, which is used by the VPT* structure, can be seen in Figure 2.9. In general, this principle also divides a set I into two partitions I_1 and I_2 . However, only one pivot x_{11} is selected from the set and the objects are divided by a radius r_1 . More specifically, if the distance between the pivot x_{11} and an object $x \in I$ is smaller or equal to the specified radius r_1 , i.e. if $d(x, x_{11}) \leq r_1$ then the object belongs to partition I_1 . Otherwise, the object is assigned to I_2 . Similarly, the algorithm is applied recursively to build a binary tree. The leaf nodes follow the same schema for addressing local buckets and remote peers.

Range Search

The $R(q, r)$ query search in both the GHT* and VPT* structures proceeds as follows. The evaluation starts by traversing the local AST of the peer which issued the query. For every inner node in the tree, we evaluate the following conditions. Having the generalized hyperplane partitioning with the inner node of format $\langle p_1, p_2 \rangle$:

$$d(p_1, q) - r \leq d(p_2, q) + r, \quad (2.1)$$

$$d(p_1, q) + r > d(p_2, q) - r. \quad (2.2)$$

For the vantage point partitioning with the inner node of format $\langle p, r_p \rangle$:

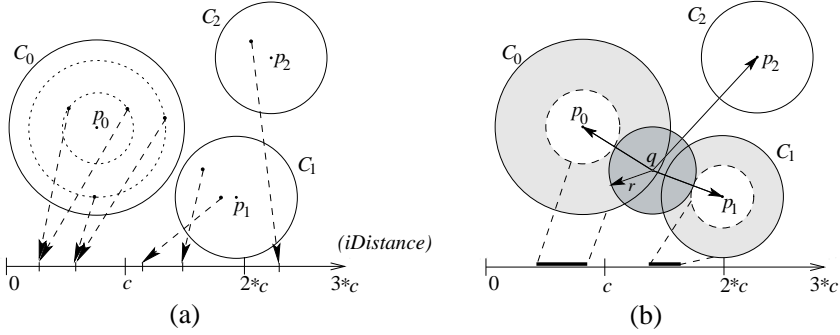
$$d(p, q) - r \leq r_p, \quad (2.3)$$

$$d(p, q) + r > r_p. \quad (2.4)$$

The right subtree of the inner node is traversed if Condition 2.1 for the GHT* or Condition 2.3 for the VPT* qualifies. The left subtree is traversed whenever Condition 2.2 or Condition 2.4 holds respectively. It is clear that both conditions may qualify at the same time for a particular range search. Therefore, multiple paths may be followed and, finally, multiple leaf nodes may be reached.

For all qualifying paths having an NNID pointer in their leaves, the query request is forwarded to identified peers until a BID pointer is found in every leaf. The range search condition is evaluated by the peers in every bucket determined by the BID pointers. All qualifying objects together form the query response set.

In order to avoid some distance computations, both the structures apply additional filtering using Equation 4.2 [p.80]. For every stored object, the distances to all pivots on the AST path from the root to the leaf with the respective bucket are held together with the data object. For example, object x_1 in Figure 2.8 has four associated numbers — the distances $d(x_1, x_{10})$, $d(x_1, x_{12})$, $d(x_1, x_8)$, $d(x_1, x_{19})$ which were evaluated during the insertion of x_1 into bucket BID_1 . In the case of VPT* structure, only half of the distances are stored, because only one pivot is present in every

Figure 2.10: The principles of *iDistance*

inner node. As is obvious, the deeper the bucket where the object is stored, the more precomputed distances are stored for that particular object and the better the effect of the filtering.

2.5.2 M-Chord

Similarly to the MCAN, the M-Chord [136] approach also transforms the original metric space. The core idea is to map the data space into a one-dimensional domain and navigate in this domain using the Chord routing protocol [167].

Specifically, this approach exploits the idea of a *vector* index method *iDistance* [88, 190], which partitions the data space into *clusters* (C_i), identifies reference points (p_i) within the clusters, and defines one-dimensional mapping of the data objects according to their distances from the cluster reference point. Having a separation constant c , the *iDistance* key for an object $x \in C_i$ is $\text{idist}(x) = d(p_i, x) + i \cdot c$.

Figure 2.10a visualizes the mapping schema. Handling a $R(q, r)$ query, the space to be searched is specified by *iDistance* intervals for such clusters that intersect the query sphere — see an example in Figure 2.10b.

This method is generalized to metric spaces in the M-Chord. No vector coordinate system can be utilized in order to partition

a general metric space, therefore, a set of M pivots p_0, \dots, p_{M-1} is selected from a sample dataset and the space is partitioned according to these pivots. The partitioning is done in a Voronoi-like manner [84] (every object is assigned to its closest pivot).

Because the *iDistance* domain is to be used as the key space for the Chord protocol, the domain is transformed by an order-preserving hash function h into M-Chord domain of size 2^m . The distribution of h is uniform on a given sample dataset. Thus, for an object $x \in C_i$, $0 \leq i < M$, the M-Chord key-assignment formula becomes:

$$m\text{-chord}(x) = h(d(p_i, x) + i \cdot c). \quad (2.5)$$

The M-Chord Structure

Having the data space mapped into the one-dimensional M-Chord domain, every active node of the system takes over the responsibility for an interval of keys. The structure of the system is formed by the Chord circle (see Subsection 2.4.2 [p.36]). This Peer-to-Peer protocol provides an efficient localization of the node responsible for a given key.

When inserting an object $x \in \mathcal{D}$ into the structure, the initiating node n_{ins} computes the $m\text{-chord}(x)$ key using Formula 2.5 and employs Chord to forward a store request to the node responsible for key $m\text{-chord}(x)$ (see Figure 2.11a).

The nodes store data in B⁺-tree storage according to their M-Chord keys. When a node reaches its storage capacity limit (or another defined condition) it requests a *split*. A new node is placed on the M-Chord circle, so that the requester's storage can be split evenly.

Range Search Algorithm

The node n_q that initiates the $R(q, r)$ query uses the *iDistance* pruning idea to choose the M-Chord intervals to be examined. The Chord protocol is then employed to reach nodes responsible for middle points of these intervals. The request is then spread to all nodes covering the particular interval (see Figure 2.11b).

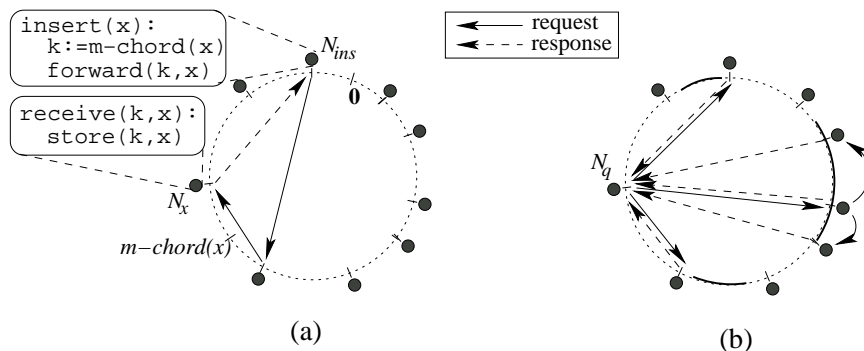


Figure 2.11: The insert (a) and range search (b)

From the metric point of view, the *iDistance* pruning technique filters out all objects $x \in C_i$ that fulfill $|d(x, p_i) - d(q, p_i)| > r$. But in M-Chord, when inserting an object x , all distances $d(x, p_i)$ have to be computed $\forall i : 0 \leq i < n$. These values are stored together with object x and the general metric filtering criterion improves the pruning of the search space.

2.6 Peer-to-Peer and Grid Computing

Another approach to distributed computing which has been very important in the past few years research is *Grid computing*.

The term Grid computing originated in the early 1990s as a metaphor for making computer power as easy to access as an electric power grid. Even if there are many definitions. In [70] Foster reports previous definition and summarize the essence of them in a three points of this checklist:

- Computing resources are not administered centrally.
- Open standards are used.
- Non-trivial quality of service is achieved.

Peer-to-Peer and *Grid computing* have a lot in common. As Foster and Iamnitchi say:

Two supposedly new approaches to distributed computing have emerged in the past few years, both claiming to address the problem of organizing large-scale computational societies: Peer-to-Peer and Grid computing. They both emerged in the past few years, both claiming to address the problem of organizing large-scale computational societies. Both approaches have seen rapid evolution, widespread deployment, successful application, considerable hype, and a certain amount of (sometimes warranted) criticism. The two technologies appear to have the same final objective: the pooling and coordinated use of large sets of distributed resources, but are based in different communities and, at least in their current designs, focus on different requirements. [71]

In the same paper they try to compare and contrast Peer-to-Peer and Grid computing. In brief, they argue that:

- both are concerned with the organization of resource sharing within virtual communities;
- both take the same general approach to solving the creation of 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 Peer-to-Peer 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.

We agree with Androutsellis-Theotokis and Spinellis that:

As Grid systems increase in scale, they begin to require solutions to issues of self-configuration, fault tolerance, and scalability, for which Peer-to-Peer research has much to offer. Peer-to-Peer systems, on the other hand, focus on dealing with instability, transient populations, fault tolerance, and self-adaptation. To date, however, Peer-to-Peer developers have worked mainly on vertically integrated applications, rather than seeking to define common protocols and standardized infrastructures for interoperability. [11]

At the end of these considerations they agree with Foster [71] about the fact that “Grid computing addresses infrastructure but not yet failure, whereas Peer-to-Peer addresses failure but not yet infrastructure”. However, as Foster in [69], they believe that, as Peer-to-Peer technologies move into more sophisticated and complex applications, (e.g. structured content distribution, desktop collaboration, and network computation), it is expected that there will be a strong convergence between Peer-to-Peer and Grid computing. The result will be a new class of technologies combining elements of both Peer-to-Peer and Grid computing, which will address scalability, self-adaptation, and failure recovery, while, at the same time, providing a persistent and standardized infrastructure for interoperability.

Chapter 3

Content-Addressable Network (CAN)

Defined in [145], Content-Addressable Network (CAN) is a DHT (see Section 2.4 [p.29]). The CAN routing algorithm provides the DHT functionality while meeting the previously enumerated design goals of scalability, efficiency, dynamicity and balanced load.

The basic operations performed on a CAN are the insertion, lookup and deletion of $(key, value)$ pairs. Each CAN node stores a chunk (called a *zone*) of the entire hash table. In addition, a node holds information about a small number of *adjacent* zones in the table. Requests (insert, lookup, or delete) for a particular *key* are routed by intermediate CAN nodes toward the CAN node whose zone contains that *key*. The CAN design is completely distributed (requiring no form of centralized control, coordination or configuration), scalable (nodes maintain only a small amount of control state that is independent of the number of nodes in the system), and fault-tolerant (nodes can route around failures). Finally the CAN design can be implemented entirely at the application level

The CAN design centers around a virtual M -dimensional Cartesian coordinate space on a M -torus. This coordinate space has no relation to any physical coordinate system. At any point in time, the entire coordinate space is dynamically partitioned among all the nodes in the system. A node learns and maintains the IP addresses

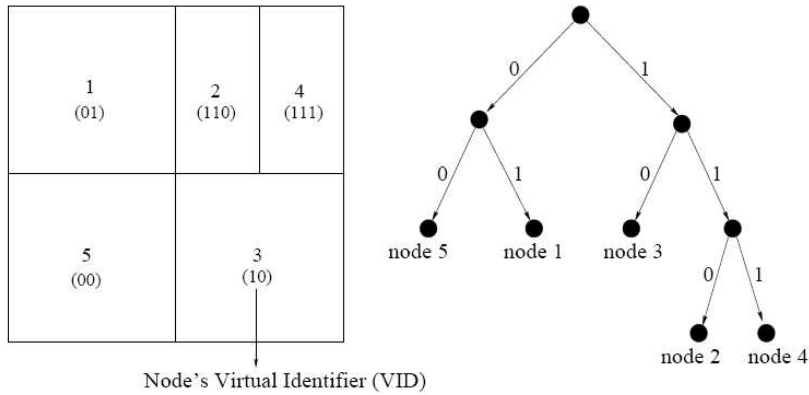


Figure 3.1: 5 nodes CAN and its corresponding partition tree.

of those nodes that hold coordinate zones adjoining its own zone. This set of immediate neighbors serve as a coordinate routing table that enables routing between arbitrary points in the coordinate space.

The Cartesian space serves as a level of indirection. The virtual coordinate space is used to store $(key, value)$ pairs by deterministically mapping the *key* into a point \hat{x} in the coordinate space using a uniform hash function. The corresponding pair is then stored at the node that owns the zone within which the point \hat{x} lies. To retrieve an entry corresponding to a given *key*, any node can apply the same deterministic hash function and retrieve the corresponding *value* from the point \hat{x} . If the point \hat{x} is not owned by the requesting node or its immediate neighbors, the request must be routed through the CAN infrastructure until it reaches the node in whose zone \hat{x} lies.

In this chapter a short introduction to the CAN is given. A further description can be found in Ratnasamy's PhD Thesis [144].

3.1 Node arrivals

Any node that joins the CAN must:

- be allocated its own portion of the coordinate space;
- discover its neighbors in the space.

In the CAN, the entire space is divided among the nodes currently in the system. The first node to join owns the entire CAN space (i.e. the complete virtual space). Each time a new node joins the CAN, an existing zone is split into two halves, one of which is assigned to the new node. The split is done by following a predefined ordering of the dimensions in deciding along which dimension a zone is to be split, so that zones can be re-merged when nodes leave. For example, for a 2-dimensional space, a zone would first be split along the first dimension, then along the second, then the first again followed by the second and so forth.

Thus, at any given step, we can think each existing zone as a leaf of a binary *partition tree*. The internal nodes in the tree represent zones that no longer exist, but were split at some previous time. The children of a tree node are the two zones into which it was split (see Figure 3.1 [144]). Labeling the edges with a predefined rule (e.g. 0 if the child zone occupies the lower half, and 1 otherwise), a zone's position (i.e. the zone's coordinate span along each dimension) in the coordinate space is completely defined by the path from the root of the partition tree to the leaf node corresponding to that zone. Every node in the CAN is addressed with a virtual identifier (VID)—the binary string representing the path from the root to the leaf node. The partition tree is not maintained as a data structure, and none of the CAN operation require a node to have knowledge of the entire partition tree. The tree is just a useful conceptual aid to understanding the structure of nodes in a CAN. Considering the tree, the allocation of a portion of the coordinate space to a node can be seen as obtaining a unique VID and the discovering of neighbors as discovering its neighbors'VIDs and IP addresses.

We can summarize the process of joining the CAN in three steps:

1. the new node must find a node already in the CAN

2. using the CAN routing mechanisms, it must find a node whose zone will be split
3. the neighbors of the split zone must be notified so that routing can include the new node.

In this chapter we will not take into consideration the bootstrap mechanism because the functioning of a CAN does not depend on the details of how this is done. For a description of how the bootstrap was implemented on the first implementation of the CAN see [144].

3.1.1 Finding a Zone

To find a node to ask for split, the joining node chooses a point \hat{x} in the space and sends a join request destined for point \hat{x} . This message is sent into the CAN via any existing CAN nodes. Each CAN node then uses the CAN routing mechanism (described later) to forward the message, until it reaches the node in whose zone \hat{x} lies.

In the CAN definition two approaches have been taken in considerations, once the owner node has been reached. The simpler just split the owner node. The other one, namely *1-hop volume check*, makes use of the fact that the owner node knows not only its own zone coordinates, but also those of its neighbors. Therefore, instead of directly splitting its own zone, the existing occupant node first compares the volume of its zone with those of its immediate neighbors in the coordinate space. The zone with the largest volume is then split. In [144] a comparison of those two approaches has been given. Results shown that *1-hop volume check* gives significantly better results. Because of the fact that *1-hop volume check* performance does depend on the average number of neighbors, increasingly the dimensionality of the CAN space the advantages of the *1-hop volume check* become more and more significantly.

3.1.2 Joining the Routing

Once the joining node has obtained its zone, it must learn the IP addresses of its coordinate neighbor set. In a M -dimensional coordinate space, two nodes are neighbors if their coordinate spans overlap along $M - 1$ dimensions and abut along one. Because a new node's zone is derived by splitting the previous occupant's zone, the new nodes' neighbor set is a subset of the previous occupant's neighbors, plus that occupant itself. Similarly, the previous occupant updates its neighbor set to eliminate those nodes that are no longer neighbors. Finally, both the new and old nodes' neighbors must be informed of this reallocation of space. Every node in the system senses an immediate update message, followed by periodic refreshes, with its currently assigned zone to all its neighbors.

The addition of a new node affects only a small number of existing nodes in a very small locality of the coordinate space. The number of neighbors a node maintains depends only on the dimensionality of the coordinate space and is independent of the total number of nodes in the system. Thus, for a M -dimensional space, node insertion affects only $O(M)$ existing nodes which is important for CANs with huge numbers of nodes. In Section 3.4 [p.70] we will further discuss load balancing in CANs.

3.2 Routing

Using its neighbor coordinate set, a node routes a message toward its destination by simple greedy forwarding to the neighbor with coordinates closest to the destination coordinates (see Figure 3.2 [144]).

For a M -dimensional space partitioned into N equal zones, individual nodes maintain $2M$ neighbors (one to advance and one to retreat along each dimension) and the average routing path length is $(M/4)(N^{1/M})$ (each dimension has $N^{1/M}$ nodes; on a torus, a destination will, on average be $(1/4)(N^{1/M})$ nodes away along each of the M dimensions). For a M -dimensional space, they can grow the number of nodes (and hence zones) without increasing per node

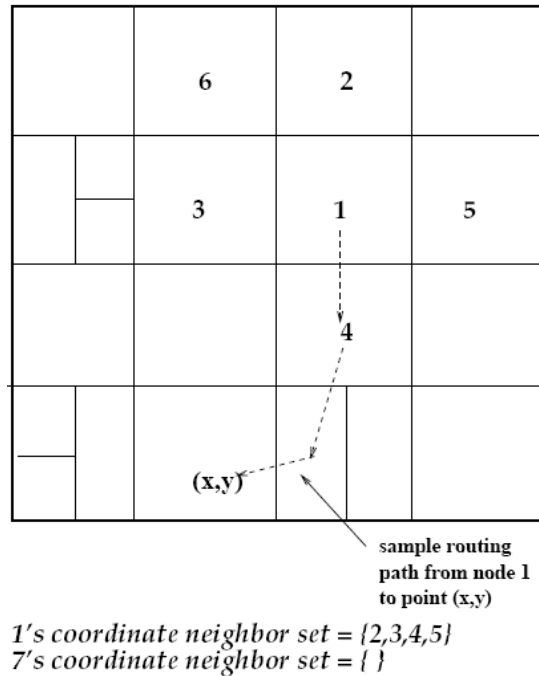


Figure 3.2: Example of routing in a 2-d space.

state while the path length grows as $O(M^{1/M})$.

Several proposed routing algorithms for location services route in $O(\log(N))$ hops with each node maintaining $O(\log(N))$ neighbors. In CAN this is possible if the number $M \geq (\log_2(N))/2$. Because of CAN is supposed to be applied to very large systems with frequent topology changes, in the consideration above M is kept fixed to keep the number of neighbors independent of the system size.

Note that, if one or more of a node's neighbors were to crash, a node would automatically route along the next best available path. If however, a node loses all its neighbors in a certain direction, and the repair mechanisms described in Section 3.3 [p.69] have not yet rebuilt the void in the coordinate space, then greedy forwarding may temporarily fail. In this case, the forwarding node first checks

with its neighbors to see whether any of them can make progress toward the destination. This *1-hop route check* is useful in circumventing certain voids and its more important at lower dimensions. If even the 1-hop route check fails, greedy routing fails and the message (see Section 3.3 [p.69]) is forwarded using the rules used to route recovery messages until it reaches a node from which greedy forwarding can resume.

3.3 Node departures

When nodes leave a CAN, the zone they occupied must be taken over by the remaining nodes. For doing this a node can explicitly hand over its zone (i.e. its own VID and its list of neighbor VIDs and IP addresses) and the associated (*key, value*) database to a specific node called the *takeover* node. If the takeover's zone can be merged with the departing node's zone to produce a valid single zone, then this is done. If not, then the takeover node can temporarily handle both zones.

The CAN is also designed to be robust to node or network failures, where one or more nodes simply become unreachable. A recovery algorithm has been defined that ensures that the takeover node and the failed node's neighbors independently work to reconstruct the routing structure at the failed node's zone. However in this case the data stored by the departing node is lost and need to be rebuilt. For example, the holders of the data can refresh the state (to prevent stale entries as well as to refresh lost entries, nodes that insert (*key, value*) into the CAN might periodically refresh these entries. Alternately, each (*key, value*) pair might be replicated at multiple points. In the CAN definition this issue is not addressed since the appropriate solution is largely dependent on application-level issues.

We do not describe the detailed recovery process by which routing state is rebuilt when a node fails. The identification of a unique node, called the *takeover* node that occupies the departed node's zone and the process by which the departed node's neighbors dis-

cover the takeover node and vice versa are full described in [144].

3.4 Load Balancing

Since data is spread across the coordinate space using a uniform hash function, the volume of a node's zone is indicative of the size of the database the node will have to store, and hence indicative of the load placed on the node. A uniform partitioning of the space is thus desirable to achieve load balancing.

Note that this is not sufficient for true load balancing because some pair (*key, value*) will be more popular than others thus putting higher load on the nodes hosting those pairs. This is basically the same as the hot-spot problem on the Web and can be addressed using caching and replication schemes as discussed in [145]. Other approaches for load balancing in CAN can be found in [162, 172, 153], while a general discussion about load balancing in DHTs can be found in [149].

3.5 M-CAN: CAN-based Multicast

The naive approach to implement flooding for a CAN overlay network is for each node that receives a message to forward that message to all of its neighbors. Nodes can filter out duplicate messages by maintaining a cache of previously received message ids. The problem with the naive strategy is that it can lead to a large number of duplicate messages.

To reduce the number of duplicates, Ratnasamy, Handley, Karp, and Shenker presented in [146] (see also [144]) an efficient flooding algorithm that exploits the structure of the CAN coordinate space to limit the directions in which each node will forward messages.

Nodes use the following five rules to decide whether to forward a message, and to decide to which neighbors to forward the message.

1. *Origin Forwarding Rule*: The multicast origin node forwards the message to all neighbors.

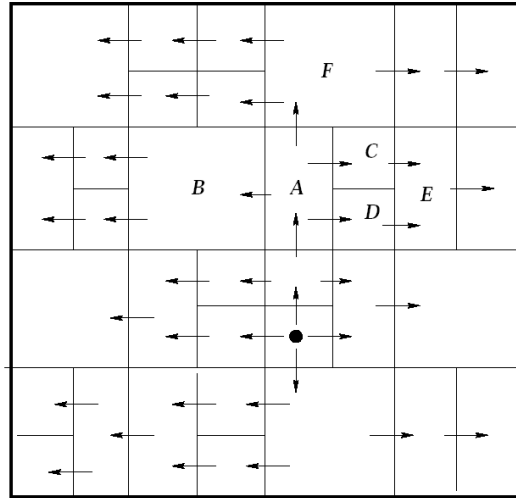


Figure 3.3: Directed Flooding over the CAN.[146]

2. *General Forwarding Rule:* A node receives a message from a neighboring node adjacent along dimension i . The node forwards that message to all adjacent neighbors along dimensions 1 through $i - 1$. The node also forwards the message to those adjacent neighbors along dimension i in the opposite direction from where it received the message.
3. *Duplicate Filter Rule:* A node caches the message-ids of all received messages. When a node receives a duplicate, it does not forward the message.
4. *Half-Way Filter Rule:* A node does not forward a message along a particular dimension if that message has already traveled at least half-way across the space from the origin coordinate in that dimension.
5. *Corner Filter Rule:* Along the lowest dimension (dimension 1), a node \underline{n} only forwards to a neighbor \underline{n}_A if a specific corner of \underline{n}_A is in contact with \underline{n} . This specific corener is

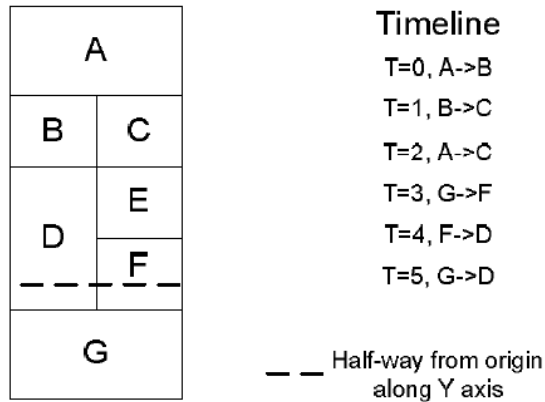


Figure 3.4: Illustration of the race condition that affects the CAN efficient flooding algorithm presented in [146].

the one \underline{n}_A that is adjacent to \underline{n} along dimension 1 and has the lowest coordinates along all other dimensions. Note that this rule eliminates certain messages that would otherwise be sent according to the two forwarding rules.

Jones, Theimer, Wang, and Wolman discovered and fixed two flaws with the above algorithm [90].

The first flaw is an ambiguity in the half-way filter rule specified above. The authors state that the above algorithm ensures there will be no duplicate messages if the CAN coordinate space is evenly partitioned (i.e. all CAN nodes have equal sized zones). The following change to the half-way filter rule is needed to ensure that this property actually holds. When deciding whether or not to forward to a neighbor \underline{n} , if \underline{n} contains the point that is halfway across the space from the source coordinate in that dimension, then we only forward to \underline{n} that neighbor from the positive direction.

The second flaw they discovered is a race condition that can lead to certain nodes never receiving the flooded message. This race condition arises because when a node receives a duplicate message, it does not forward that message. Therefore, the order in which a node receives a message from its neighbors may determine the

directions in which that message is forwarded.

To demonstrate this problem, Figure 3.4 [90] illustrates a situation where one of the nodes does not receive the multicast message. This figure shows a small portion of a 2-dimensional CAN, where the dashed line in the figure is the location along the y axis that is half-way from the origin. The sequence of message delivery times listed in the time line portion of Figure 3.4 causes node E to never receive the message. Note that a different ordering of message reception either at node \underline{n}_C or at node \underline{n}_D would have led to proper message delivery at node \underline{n}_E . For example, if we switch the order of messages at times $T=1$ and $T=2$, then the message from \underline{n}_A to \underline{n}_C is delivered before the message from \underline{n}_B to \underline{n}_C , which means that node \underline{n}_C will forward the message to \underline{n}_E . The idea behind fixing [90] the flooding algorithm is to make static forwarding decisions based on the relative position of a node to the multicast origin, rather than dynamic forwarding decisions based on the order of incoming messages. The new algorithm presented in [90] breaks up the forwarding process into two stages. In first stage, a node decides which dimensions and directions to forward message along. In the second stage, a node applies a second set of rules to filter the subset of neighbors that satisfy the first stage rules.

The stage one forwarding rules presented in [90] are:

1. If a node's region overlaps the origin along all dimensions less than or equal to i , then this node will forward the message in both the positive and the negative directions along dimension i .
2. If a node's region overlaps the origin along all dimensions less than i , then this node will forward the message only in one direction along dimension i . The direction to forward the message will be away from the origin coordinate, toward the half-way point.
3. For the lowest dimension (dimension 1), always forward only in one direction. As before, the forwarding direction will be away from the origin coordinate, toward the half-way point.

The stage two filtering rules presented in [90] are:

1. For all dimensions greater than 1, only forward to a neighboring node along dimension i if that neighbor's region overlaps the origin coordinates for all dimensions less than i .
2. The half-way filter rule from the original algorithm, with our modification described above.
3. The corner filter rule from the original algorithm.

Although the rules for this algorithm look somewhat different from the original algorithm, the way that messages flow through the CAN coordinate space is quite similar to the original algorithm. An important side effect of the modified flooding algorithm is a significant reduction in the number of duplicate messages, due to the first rule in the filtering stage of the new algorithm.

In [43] the tree-based application-level multicast for DHTs first introduced in Pastry¹ (namely SCRIBE[42]) and Tapestry² (namely Bayeux [195]), has been adapted to operate on CAN.

A delay analysis of the [146] and [43] approaches has been given in [82]. A good survey of application-level multicast approaches in which the M-CAN is compared with SCRIBE can be found in [95].

3.6 CAN Related Works

In [12] Range queries over CAN have been investigated. They proposed three simple strategies for propagating Range query requests, and strategies to minimize the communication overhead during the attribute updates. Their strategies are limited to attributes that have a single value which belongs to \mathbb{R} .

Nearest Neighbor queries have been studied in [33]. They supposed that the hash function is able to map near objects into near keys. Moreover, they only consider the distance between the objects as the distance between the keys.

¹we described Pastry in Subsection 2.4.3 [p.39]

²we described Tapestry in Subsection 2.4.4 [p.41]

In [147] various types of attacks on the CAN and possible countermeasures have been proposed. However, the problem of peer inserting unwanted or unpopular data into the CAN is not considered in this paper.

Voice over IP (VoIP) over DHTs is considered in [16]. In particular, they presented a robust architecture for Session Initiation Protocol (SIP) infrastructures called CAN-based SIP (CASIP) which makes use of the CAN

Chapter 4

MCAN

In this chapter, we present the Metric Content-Addressable Network (MCAN) which is the main object of this thesis. MCAN, originally presented in [65], is an extension of the CAN¹ to support storage and retrieval of generic metric space² objects .

In order to manage metric data, the MCAN uses a pivot-based technique, presented in Section 4.1 [p.78], that maps metric data objects $x \in \mathcal{D}$ to an M -dimensional vector space \mathbb{R}^M . Then, the CAN protocol is used to partition the space and for navigation.

Because of a particular property of the chosen mapping, namely *contractivness*, during similarity queries execution it is always possible to evaluate a lower bound for the distance between a generic object and all the objects stored in a generic node. Thus it is possible to exclude peers from the query execution.

In order to reduce the number of the distances evaluated by each involved node, MCAN uses the pivot-based filtering described Section 4.2 [p.80].

In Section 4.8 [p.86] Range query similarity search over the MCAN, defined in [65], is presented. Moreover, three algorithms for executing Nearest Neighbor queries over the MCAN are presented in Section 4.9 [p.88] (they were defined in [66]).

¹see Chapter 3 [p.63]

²see Section 1.3 [p.7]

4.1 Mapping

In MCAN a special mapping function is used to map objects from a generic metric space \mathcal{D} to an M -dimensional vector space \mathbb{R}^M in which we use L_∞ as distance (see Definition 1.3.1 [p.10]). We will prove that this mapping is a contraction mapping. In the following we give a definition of mapping function (Definition 1) followed by the definition of contraction mapping (Definition 2) and by the proof that our mapping function is a contraction mapping (Theorem 1).

Definition 1. Let $\{p_1, p_2, \dots, p_M\}$ a set of preselected objects in \mathcal{D} , called pivots, and x a generic object in \mathcal{D} . We define the mapping function F as:

$$F(x) : \mathcal{D} \rightarrow \mathbb{R}^M = (d(x, p_1), d(x, p_2), \dots, d(x, p_M))$$

Using F (to map objects from \mathcal{D} in \mathbb{R}^M) and L_∞ as metric distance for the in \mathbb{R}^M we can map any objects from the original metric space $\mathcal{M}(\mathcal{D}, d)$ in a new vector space, which is also metric, $\mathcal{M}_M(\mathbb{R}^M, L_\infty)$.

We now define an important subclass of mapping functions: *contraction*.

Definition 2. A *contraction mapping*, or *contraction*, is a function f from a metric space $\mathcal{M}(\mathcal{D}, d)$ to another metric space $\hat{\mathcal{M}}(\hat{\mathcal{D}}, \hat{d})$, with the property that there is some real number $0 \leq \lambda \leq 1$ for which:

$$\forall x, y \in \mathcal{D}, \quad \hat{d}(f(x), f(y)) \leq \lambda d(x, y) .$$

If $\lambda \in [0, 1)$ then the mapping is said to be a *strict contraction* while if $\lambda = 1$ then the mapping is said to be *nonexpansive* or simply *contraction*.

Sometimes *strict contractions* are called just *contractions* in which case the $\lambda = 1$ case is always referred as *nonexpansive*.

Now we prove that the proposed mapping is a contraction mapping.

Theoreme 1. *F is a contraction mapping from a generic metric space $\mathcal{M} = (\mathcal{D}, d)$ to the $\mathcal{M}_M = (\mathbb{R}^M, L_\infty)$ metric space, (i.e. for any pair of objects in \mathcal{D} , the distance between the mapped objects in the derived space \mathcal{M}_M obtained using F is never larger than the distance in the original metric space \mathcal{M}).*

Proof. Let $x, y \in \mathcal{D}$ a pair of objects and $\hat{x}, \hat{y} \in \mathbb{R}^M$ their mapped values, i.e. $\hat{x} = F(x)$ and $\hat{y} = F(y)$. From the L_∞ and F definitions:

$$L_\infty(\hat{x}, \hat{y}) = \max_i |d(x, p_i) - d(y, p_i)| .$$

Since the triangle inequality (Equation 1.5) for d in the original metric space \mathcal{M} holds

$$\forall i, \quad |d(x, p_i) - d(y, p_i)| \leq d(x, y)$$

then

$$L_\infty(\hat{x}, \hat{y}) = \max_i |d(x, p_i) - d(y, p_i)| \leq d(x, y)$$

□

Finally we proved that our mapping function F is a contraction. In other words the distance L_∞ between any pair of mapped objects in the derived \mathcal{M}_M vector space is a lower bound of the “original” metric distance d between the two objects in \mathcal{M} . This properties, together with the object distributing algorithm inherits from the CAN, will permit excluding some peers from the similarity query execution without lost of results.

4.1.1 Pivot Selection

Using pivot for mapping, as we do, what we would the mapping to be less contractive as possible. To select convenient pivots in our experiments we used the *incremental selection technique* originally proposed in [35] together with other pivot selection techniques. They showed that the incremental selection is the best method in practice. Basically, the algorithm tries to maximize the average L_∞ distance between two objects

Also, they showed that good pivots have the characteristic to be outliers, that is, good pivots are objects far away from each other and from the rest of the objects of the database, but an outlier does not always have the property to be a good pivot. It is interesting to note that outliers sets have good performance in uniformly distributed vector spaces, but have bad performance in general metric spaces, even worse than random selection in some cases.

4.2 Filtering

In our experiments we used pivoted filtering to reduce distance evaluations during the execution of Range and Nearest Neighbor similarity queries by an involved nodes. However, a generic node participating in MCAN could use its own centralized data structures for efficiently searching between its metric objects (see Section 1.4 [p.12] for a small survey of centralized access methods for metric spaces).

Performing a range query $R(q,r)$ (Subsection 1.2.1 [p.4]) we search for objects $x \in \mathcal{X}$ (where $\mathcal{X} \subseteq \mathcal{D}$ are the stored objects) that are nearest than r to the q , i.e.:

$$d(x, q) \leq r ,$$

we compute for the query object $q \in \mathcal{D}$, it's mapped object where

$$\hat{q} = F(q) = (d(q, p_1), \dots, d(q, p_M)) .$$

Because the mapping is a contraction $L_\infty(\hat{x}, \hat{q}) \leq d(x, q)$ (see Theorem 1). Thus we can avoid the evaluation of $d(x, \hat{q})$ whenever

$$L_\infty(\hat{x}, \hat{q}) > r . \tag{4.1}$$

In other words, the object x can be discarded if there exists a pivot p_i such that

$$\exists i, \quad |d(q, p_i) - d(x, p_i)| > r. \tag{4.2}$$

During the evaluation of a k Nearest Neighbors $kNN(q)$ (Section 1.2.2 [p.4]) if $y_1, \dots, y_k \in \mathcal{D}$ are the temporary results, we can avoid the evaluation of $d(q, x)$ if

$$L_\infty(\hat{x}, \hat{q}) > d(y_k, q) . \quad (4.3)$$

In other words, an object x can be discarded if there exists a pivot p_i such that

$$\exists i, \quad |d(q, p_i) - d(x, p_i)| > d(y_k, q) . \quad (4.4)$$

While Equation 4.1 and Equation 4.3 are specific to MCAN mapping, Equation 4.2 and Equation 4.4 describe essentially the same conditions in the form used in the pre-computed distances literature (see Subsection 1.4.3 [p.13]).

Here we will not consider specific data structures to efficiently execute pivoted filtering. In our experiments we will consider only distance evaluation cost thus reducing the importance of specific data structures for optimizing memory or disk usage. However, well-known data structures have been proposed in the literature to efficiently exploit pre-computed distances, e.g. LAESA [122] and Spaghettis [44] (see Subsection 1.4.3 [p.13]).

4.3 Regions

We denote a peer of MCAN by the bold symbol \underline{n} . Each peer \underline{n} maintains its region information referred as $\underline{n}.R$. Moreover, since the region $\underline{n}.R$ is an hyper-rectangle it can be uniquely identified by its vertex closest to the origin, denoted as

$$\underline{n}.R.\hat{v} = (\underline{n}.R.v_1, \underline{n}.R.v_2, \dots, \underline{n}.R.v_M) ,$$

and by the lengths of the relative sides, i.e.

$$\underline{n}.R.l_1, \underline{n}.R.l_2, \dots, \underline{n}.R.l_M .$$

More precisely, the region $\underline{n}.R$ is defined as follows

$$\underline{n}.R = \{\forall \hat{x} \in \mathbb{R}^M \mid \forall i, \underline{n}.v_i \leq x_i < \underline{n}.v_i + \underline{n}.l_i\}$$

The peer \underline{n} also maintains the set of the neighbor peers' information $\underline{n}.M \subset \{\underline{n}_1, \dots, \underline{n}_h\}$.

We can now introduce the formal definition of an M -dimensional MCAN structure, referred as MCAN^M , which is composed of a set of N ($N > 0$) network peers $\{\underline{n}_1, \dots, \underline{n}_h\}$ such as:

$$\forall i, j \mid i \neq j \quad \underline{n}_i.R \cap \underline{n}_j.R = \emptyset, \quad (4.5)$$

$$\bigcup_{i=1}^N \underline{n}_i.R = R^M, \quad (4.6)$$

$$\begin{aligned} \underline{n}_i \in \underline{n}_j.M \Leftrightarrow \exists k \mid \\ (\underline{n}_i.R.v_k + \underline{n}_i.R.l_k = \underline{n}_j.R.v_k) \vee (\underline{n}_j.R.v_k + \underline{n}_j.R.l_k = \underline{n}_i.R.v_k), \\ \forall w \neq k \mid \underline{n}_i.R.v_w, \underline{n}_i.R.v_w + \underline{n}_i.R.l_w \cap \\ \cap [\underline{n}_j.R.v_w, \underline{n}_j.R.v_w + \underline{n}_j.R.l_w] \neq \emptyset. \end{aligned} \quad (4.7)$$

Equation 4.5 states that the zones covered by the network peers do not overlap. Equation 4.6 states that the union of the zones covers the whole MCAN^M space R^M (there are no holes). Finally, Equation 4.7 declares the condition for a network node \underline{n}_i to be a neighbor of \underline{n}_j .

4.4 Construction

An important feature of the CAN structure is its capability to dynamically adapt to data-set size changes. As we will see in the experimental evaluation, we are interested in preserving the scalability of the MCAN, which means that we want to maintain a stable the response time of query execution. Since the number of objects a peer can maintain is limited, when a peer exceeds its limit it splits by sending a subset of its objects to a free peer that takes responsibility for a part of the original region. Note that, limiting the number of objects each peer can maintain, we also limit (reduce)

the number of distance computations a peer have to compute during a query evaluation.

It is important to observe that in some cases we might want to use all the peers available in the network. Previous work like have studied this possibility in a generic CAN structure by allowing a peer to split even if it does not exceed its storage capacity. Obviously, such methodology can also be applied in our MCAN. On the other hand, in a Peer-to-Peer environment, we would like to let the peers the possibility to freely join and leave the network, without affecting its consistency. As showed in Chapter 3 [p.63], this is possible with a CAN, which even provides some fault-tolerance capabilities.

Since pivots need be determined before the insertion starts, we assume a characteristic subset of the indexed dataset (about 5000 objects) is known at the beginning. For selecting the pivots, we use the Incremental Selection algorithm described in Subsection 4.1.1 [p.79]. This algorithm tries to maximize the average distance L_∞ between two arbitrary objects in the derived space (i.e. $L_\infty(F(X), F(Y))$).

4.5 Insert

An insert operation can be initiated in any peer of the MCAN. It starts by mapping the inserted object X to the virtual coordinate space using function $F()$, and proceeds by checking if $\hat{x} = F(X)$ lies in the zone maintained by the peer \underline{n} , i.e. $\hat{x} \in \underline{n}.R$. If this is not the case, the peer forwards the insertion request. From this point, the insertion proceeds with the greedy routing algorithm used in standard CAN structures: the inserting peer forwards the insertion operation to the neighbor peer which is closer to the point \hat{x} by using the L_∞ distance. The objective is to find the peer \underline{n} for which $\hat{x} \in \underline{n}.R$, minimizing the number of messages. We refer to this special peer as $\mu(\hat{q})$ (i.e. $\hat{q} \in \mu(\hat{q}).R$). If \hat{x} lies in the region maintained by the receiving peer, the object X is stored there, otherwise a neighbor peer is selected with the same technique

and the insert operation is forwarded again until the object X is inserted.

The peer $\mu(\hat{x})$, which stores the object X must reply to the peer that started the insert operation. If the peer $\mu(\hat{x})$ exceeds its capacity it splits. Eventually, the object X is inserted in $\mu(\hat{x})$ or in the new allocated peer.

4.6 Split

In MCAN, we apply a balanced split, i.e. the resulting regions contain the same number of objects. During this process, the splitting peer just requests a peer from a free peer list to join the network, and one half of the metric objects is then reallocated there.

If we define \underline{n}_1 as the splitting peer, $\underline{n}_1.R$ as the old region, $\underline{n}_1.R'$ as the new one, and \underline{n}_2 as the new peer, the split regions must satisfy the following equations:

$$\begin{cases} \underline{n}_1.R' \cup \underline{n}_2.R = \underline{n}_1.R \\ \underline{n}_1.R' \cap \underline{n}_2.R = \emptyset \end{cases}$$

Moreover, to respect these constraints, we create the new two regions by dividing the original one along one coordinate of the space. Therefore, the new regions, $\underline{n}_1.R'$ and $\underline{n}_2.R$, must satisfy the following equations:

$$\begin{cases} \underline{n}_1.R'.v_s = \underline{n}_1.R.v_s \\ \underline{n}_2.R.v_s = \underline{n}_1.R'.v_s + \underline{n}_1.R'.l_s \\ \underline{n}_2.R.l_s = \underline{n}_1.R.l_s - \underline{n}_1.R'.l_s \end{cases}$$

Note that that we only have to choose s and $\underline{n}_1.R'.l_s$. In order to decide s , for each dimension i we find $\underline{n}_1.R'.l_i$ that divide the objects into two halves. Moreover, we choose to split along the dimension that maximizes the length of the shortest side.

After the splitting process, the peer \underline{n}_1 sends a message to all its neighbors $\underline{n}_1.M$ informing them about the update of its region. It also sends information about the new peer to the neighbors that are also neighbors of \underline{n}_2 . The new peer is informed by \underline{n}_1 about

its neighbors $\underline{n}_2.M$ (note that $\underline{n}_2.M \subseteq \underline{n}_1.M$). At the end, \underline{n}_1 can discard information about the peers that are not more its neighbors.

4.7 Execution End Detection

One of the problem of Peer-to-Peer systems able to perform similarity search such as Range and Nearest Neighbor queries is that typically a set of nodes is involved in the execution to the query and all of them return their results to the requester. Moreover, we can not expected the answers to come in the order in which nodes are involved. However, the requester must be able to exactly know when there are no more nodes to wait for. Obviously, we don't want to force the requester to directly contact each involved node. In fact, because of the partial information about the network the requester has, it is not able to know in advance which nodes will participate in the query process.

When the requester starts an operation, if none of its neighbor must be involved, it forwards its request to a not involved node. The routing mechanism of the CAN underlying structure, will forward the request to an involved node. Thus there is always an involved node which was contacted either directly from the requester or from a forwarding peer (not from another involved node). We call this special node the first-involved node. Without lost of generality, this first-involved node could even be the requester itself. After a first-involved node is reached, using the routing mechanism, all other involved nodes are contacted by an involved node (see Range query in Section 4.8 [p.86] and Nearest Neighbor in Section 4.9 [p.88]).

Our simple solution for similarity queries execution end detection is based on a list of involved nodes managed by the requester. Any answering node must communicate the nodes that it has asked to participate in the similarity query execution. This node are added to the involved nodes list by the requester. An query execution is ended when the first-involved node and all the nodes which are in the involved nodes list have answered.

When this condition is satisfied, it does not exists an involved

node which has not yet answered. In fact, this hypothetical node would have been involved by another node which did not answer yet and which is not yet in the involved node list, and so on. Recursively following this chain of involved nodes, we should be able to find the first-involved node. But this is not possible because the first-involved node is already answered. To better understand, it is possible to imagine a tree which has the first involved node as root and involved nodes as children of the node who involved them. From the root it must be possible to reach any involved node and vice versa.

4.8 Range Query

A range query operation $R(q,r)$ can start from any MCAN node. As shown in Figure 4.1, for a given query object and range radius, there is a certain number of nodes whose regions intersect the query region which is a hypercube with \hat{q} as center and $2r$ as side length. We denote as $\langle \hat{q}, r \rangle$. Obviously, only the intersecting nodes must process the range query operation. The requesting node maps the query object into the virtual coordinate space using the function $F()$. Then it checks if it is involved in the range query operation (i.e. when it intersects $\langle \hat{q}, r \rangle$). If the node is not involved in the query, it forwards the range query operation to the neighbor node that is closest to region $\langle \hat{q}, r \rangle$, using the L^∞ distance. This operation is performed in a similar way as described for the insert operation.

When a node that is involved in the range query is reached by the query request, it forwards it to each neighbor that is also involved and then it starts processing the range query over its local data-set. From this point the multicast algorithm proposed in [146] with improvement and corrections described in [90] (see Section 3.5 [p.70]) is used to forward the request to all involved peers. To efficiently parallelize the operation, each peer forwards the message before starting executing the $R(q,r)$ locally.

In this thesis, we used a superset of the pivots chosen to define the MCAN space to reduce the number of distance evaluations

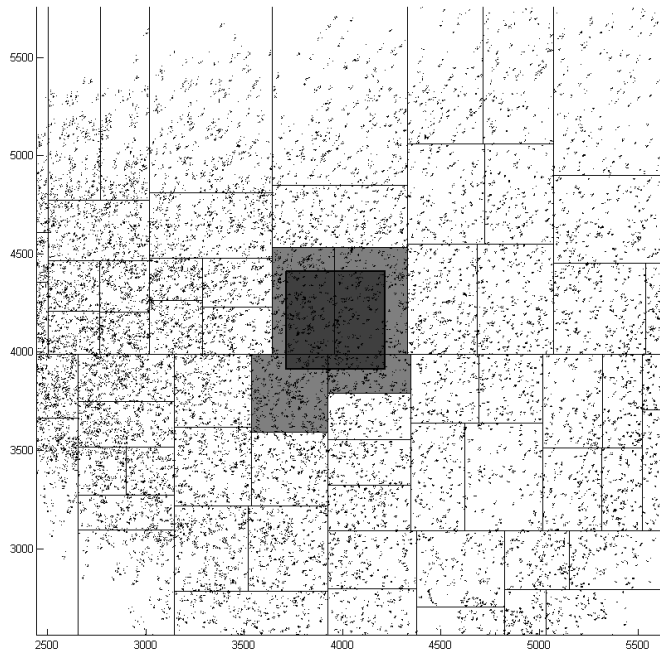


Figure 4.1: Example of Range query in a two dimensional space. The darker square is the query region, while the brighter rectangles correspond to the involved nodes.

performed inside a single node. Using the pivot-based filtering (see Section 4.2 [p.80]), we are able to significantly reduce the number of distance evaluations inside the nodes. In a more sophisticated implementation of MCAN, each node could have its own local data structure to efficiently search inside a single node.

In order to allow the requesting node to know when all the nodes

The requester, which is the peer receiving the answers coming from the involved nodes, detects the execution end using the mechanism described in Section 4.7 [p.85].

4.9 Nearest Neighbor query

In MCAN, we have developed three different strategies to perform kNN queries [66]: Parallel Execution (PE), Sequential Execution (SE), and Mixed Mode Execution (MME). Each of these techniques has its advantages and disadvantages which will be discussed later.

All these three strategies start by locating the peer that contains the query object q . We refer to this special peer as $\mu(\hat{q})$. The location of $\mu(\hat{q})$ is performed exactly the same way as for the insertion operation described in Section 4.5 [p.83]. The kNN proceeds in the peer $\mu(\hat{q})$ and finds the k objects nearest to q . Note that, we assume that there are at least k objects in $\mu(\hat{q})$. Because of the splitting rule this condition is guaranteed for any k less or equal to half of the peer capacity. However, in case k is greater than the number of objects contained in $\mu(\hat{q})$, the algorithms could be easily modified by forwarding the kNN request to the most promising peer until the temporary result list contains k objects. Therefore, the first k objects, i.e. the objects with the shortest distances to q , are the candidates for the kNN result. However, there may be other objects in different peers' regions that are closer to the query than some of those k candidates. Nevertheless, since the MCAN space is contractive, these objects are within the distance to the k -th objects found to the query. In order to verify if there are other peers involved in the query, $\mu(\hat{q})$ controls if the hypercube $\langle \hat{q}, d(x_k, q) \rangle$ is completely contained in $\mu(\hat{q}).R$ (where x_k is the k -th element of the candidate result set of the kNN). If this is true, the kNN search correctly terminates and the k objects retrieved by $\mu(\hat{q})$ represent the result of the kNN query.

When $\langle \hat{q}, d(x_k, q) \rangle$ is not completely contained in the region $\mu(\hat{q}).R$, we have to check if there are other peers that maintain objects near to q , respecting the k objects found in the peer $\mu(\hat{q})$. From this stage, our three proposed algorithms start working differently. In particular, the algorithms differ in the way they propagate the kNN query execution among the involved peers. The generic behavior of these three approaches can be characterized as follows:

PE All the peers overlapping the hypercube $\langle \hat{q}, d(x_k, q) \rangle$ are in-

volved in the kNN operation. The overlapping peers that receive the query first forward the kNN query and only then they start evaluating the query on their local data.

SE The $\mu(\hat{q})$ peer only involves the most near neighbor to q . This peer first computes locally the kNN query updating its temporary result list and only after this involves the next peer most near to q , is needed.

MME The $\mu(\hat{q})$ peer involves its neighbors that overlap the hypercube $\langle \hat{q}, d(x_k, q) \rangle$. Every peer first computes locally the kNN query updating its temporary results list and only after this involves its neighbors that overlap the updated hypercube $\langle \hat{q}, d(x_k, q) \rangle$.

The query propagation of PE requires an application level multicast. In fact, starting from $\mu(\hat{q})$, the query is forwarded to all peers which overlap the hypercube $\langle \hat{q}, d(x_k, q) \rangle$. The multicast algorithm proposed in [146] with improvement and corrections described in [90] (see Section 3.5 [p.70]) are used in MCAN in order to reduce the number of replicated messages. The same algorithm is also used by MME. Actually, in MME the query propagates as in the PE except for the fact that the hypercube can reduce its size during the query propagation.

It is important to note that the three algorithms differ not only in the temporal sequence in which the peers are involved but they also differ in the number of accessed peers. In fact, the algorithms MME and SE can take advantage of the partial kNN evaluation for optimizing the query by possibly reducing the number of peers which the kNN query must be forwarded to. This optimization cannot be exploited in PE and it is optimum for the SE algorithm. On the other hand, while the parallelization of the kNN operation is maximum for the PE, for SE there is no parallelization at all. The third approach (MME) represents a trade off between PE and SE strategies.

The three algorithms also differ in terms of the total number of distance computations. In fact, during the kNN query forwarding,

in all the three approaches, the peers send along with k and the query object q a list of the distances of the current candidate result set of nearest neighbors. More precisely, a peer \underline{n} which evaluates the kNN updates the ordered list L^k defined as:

$$L^k(i) = d(X_i, q),$$

where the object X_i belongs to the merged result set of the kNN query evaluated both by \underline{n} and by a certain number of peers (it depends on the algorithm) which have been involved.

Concerning the PE approach, L^k is sent by the peer $\mu(\hat{q})$ to all its neighbors involved in the kNN query (if any). This information is then forwarded (unmodified) to the other peers involved in the kNN query. L^k together with the pivot objects can be exploited by a peer in order to reduce the number of distance computations during the kNN evaluation exploiting the filtering described in Section 4.2 [p.80]. It is clear that in PE approach L^k cannot be updated because the peers first forward the query to their neighbors and then they proceed with the query evaluation. On the contrary, in the MME approach the peers can produce and forward a more accurate version of L^k with the advantage of being able to reduce both the number of peers involved and the number of distance computations with respect to PE. In fact, the number of peers involved in the query and the number of distance computations both depend on the distance $d(x_k, q)$ which typically decrease as the kNN computation proceeds. Finally, the query evaluation before the forwarding in MME reduces the degree of parallelism.

The SE algorithm takes the maximum advantage of information stored in L^k . The peers are involved one after the other according to their region distance from q , and each peer evaluates the query before forwarding it to the next peer. For this reason, the peer that receives the forward of the kNN must know the current list of the peers that could be involved in the query. This list consists of the set of the not yet involved peers whose distances (of their regions) from q are less than or equal to $d(x_k, q)$ and that are neighbors of previously involved peers. This is necessary since the neighbor of a peer that is involved in the kNN needs not be a neighbor of the

```

receive  $q, L^k$ 
 $d_k := L^k(k)$ ; #  $d_k$  is  $d(x_k, q)$ 
 $N := \{\forall \mathbf{m} \in \underline{n}.M \mid \langle \hat{q}, d_k \rangle \cap \mathbf{m} \neq \emptyset\}$ ;
if  $L^k = \emptyset$  then
    # the peer is  $\mu(\hat{q})$ :
     $(L^k, \mathcal{A}) := \text{searchkNN\_local}(q, L^k)$ ;
    for each  $\mathbf{m} \in N$ 
        send  $q, L^k$  to  $\mathbf{m}$ ;
    end for each
else
    for each  $\mathbf{m} \in N$ 
        send  $L^k$  to  $\mathbf{m}$ ;
    end for each
     $(L^k, \mathcal{A}) := \text{searchkNN\_local}(q, L^k)$ ;
end if
send  $\mathcal{A}$  to the requesting peer

```

Figure 4.2: PE algorithm

next peer involved in the SE sequence. Note that at the end of the kNN computations (performed by each peer) this list is pruned by removing the peers whose distances from q become greater than the actual value of $d(x_k, q)$. When the list is empty, the operation terminates and the result is sent to the requesting peer.

In general, we can consider two aspects of the kNN operation costs: its parallelism, which is necessary for the scalability, and its total computational cost. The PE approach tries to maximize parallelism while the SE tries to minimize the total computational costs. MME is somewhere in the middle.

In Figures 4.2, 4.3, and 4.4, we sketch the algorithms of the approaches PE, MME, and SE, respectively. As can be seen in the sketches, MCAN does not make use of a coordinating peer. Any peer sends its result set to the requesting peer (i.e. the peer which started the kNN operation). The requesting peer merges the result lists coming from the involved peers. Note that, in the algorithms we assume the distances between the results and the query are sent

```

receive  $q, L^k$ 
 $(L^k, \mathcal{A}) := \text{searchkNN\_local}(q, L^k);$ 
 $d_k := L^k(k); \# d_k \text{ is } d(x_k, q)$ 
 $N := \{\forall \mathbf{m} \in \underline{n}.M \mid \langle \hat{q}, d_k \rangle \cap \mathbf{m} \neq \emptyset\};$ 
for each  $\mathbf{m} \in N$ 
    send  $q, L^k$  to  $\mathbf{m};$ 
end if
send  $\mathcal{A}$  to the requesting peer

```

Figure 4.3: MME algorithm

```

receive  $q, L^k, N$ 
 $(L^k, \mathcal{A}) := \text{searchkNN\_local}(q, L^k);$ 
 $N := N + \underline{n}.M; \# \text{ adds the neighbors to the list}$ 
 $p := \text{GetNearestPeer}(N, q);$ 
 $\# \text{ select peers that will not be involved}$ 
 $T := \text{GetPeersFartherAwayThan}(N, q, d_k);$ 
 $N := N - T - \{p\};$ 
send  $q, L^k, N$  to  $\mathbf{p};$ 
send  $\mathcal{A}$  to the requesting peer

```

Figure 4.4: SE algorithm


```

function ( $L^k, \mathcal{A}$ ) =searchkNN_local( $q, L^k$ )
note:  $\underline{n}$  is the current peer and  $P_1, \dots, P_w$  the pivots of
the MCAN
 $T^k := 0$  # is a temporary list of objects and distances
for  $i$  from 1 to  $k$ 
    # for results belonging to other peers we just have
    distances
     $T^k(i) := (\text{null}, L^k(i));$ 
end for
for each  $X \in \underline{n}$  # where  $\underline{n}$  is the current peer
    # all these distances were pre-evaluated
    if  $|d(q, P_i) - d(X, P_i)| \leq d(L^k(k), q)$  then
        #  $d(X, q)$  is not pre-evaluated
        if  $d(X, q) < d(L^k(k), q)$  then
             $T^k := T^k - \{T^k(k)\} + \{X, d(X, q)\};$  # pre-
            servicing order
        end if
    end if
end for each
 $\mathcal{A} := \emptyset;$ 
for  $i$  from 1 to  $k$ 
     $L^k(i) := T^k(i).\text{distance};$  # for next peers we need
    only distances
     $\mathcal{A} := \mathcal{A} + T^k(i).\text{object};$  # object can be null
end for
end function

```

Figure 4.5: Implementation of the function searchkNN_local

together with the objects `id`, even if not reported in the algorithms. In Figure 4.5, we report the *searchkNN_local* used by the previous algorithms.

In Section 4.7 [p.85], we described how the requesting peer realizes when the range query operation has terminated. The application level multicast algorithm (see Section 3.5 [p.70]) is not reported in the algorithm sketches.

An interesting direction of investigation is to generalize the *kNN* algorithms by parameterizing their behavior. Let $\alpha, \beta \in [0, 1]$ be the parameters for this new algorithm and $\beta > \alpha$. A peer performing the *kNN* first involves its neighbors whose regions overlaps $\langle \hat{q}, \alpha d(x_k, q) \rangle$. After the local execution of the *kNN* query the peer involves both the not yet involved neighbor that is closest to the query \hat{q} and those neighbors whose regions overlaps $\langle \hat{q}, \beta d(x_k, q) \rangle$. Note that for SE $\alpha, \beta = 0$, for MME $\alpha = 0, \beta = 1$, for PE $\alpha = 1$ and β is useless because all the neighbors are involved before the local execution of the *kNN*.

Chapter 5

MCAN Evaluation

In this chapter we report the results of an extensive experimental evaluation of MCAN performance with particular emphasis on scalability with respect to the dataset size. All the experiments were conducted on a real Java implementation of the MCAN over a high-speed LAN communicating via the TCP and UDP protocols.

In Section 5.1 we report the experimental results obtained varying the dimensionality of the mapped space. In Section 5.2 we report the experimental results obtained for Range queries. These results were originally reported in [65]. Finally, in Section 5.3, we report the experimental results obtained by the the Nearest Neighbor algorithms described in Section 4.9, which were originally reported in [65].

5.1 Dimensionality of the Mapped Space

As defined in Section 4.1 [p.78], in MCAN we make use of a mapping function F to map objects from a generic metric space \mathcal{D} to an M -dimensional vector space \mathbb{R}^M . Choosing M is a trade-off between three performance issues. Because M is the dimensionality of the space partitioned using the CAN (see Chapter 3 [p.63]), it directly affects both the average number of neighbors per node and the average routing path length. Moreover, because similarity queries in MCAN, as in any distributed similarity search structures,

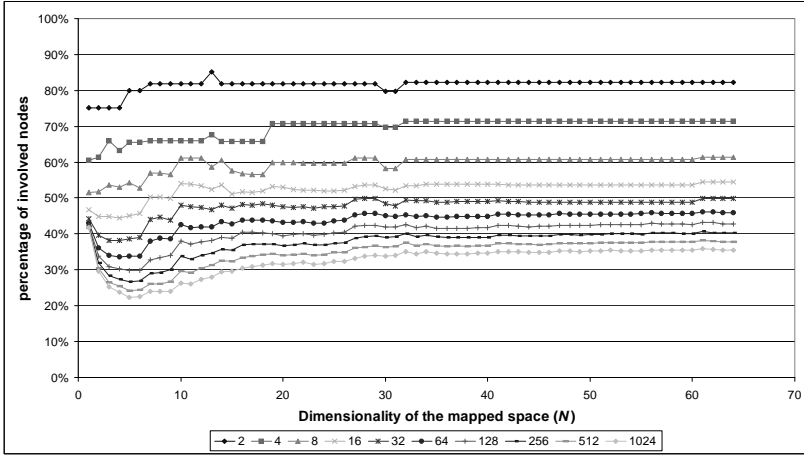


Figure 5.1: Average percentage of involved nodes for increasing M performing range queries with $r = 1000$ and various number of nodes

typically involve more than one node, M is also correlated with the percentage of involved nodes.

All the experiments of this section were conducted on 1,000,000 object dataset of 45-dimensional *vectors* of extracted color image features. The similarity of the vectors was measured by a *quadratic-form distance* [158]. The distribution of the dataset is quite uniform and such a high-dimensional data space is extremely sparse. The same dataset will be also used, together with other datasets, for Range queries (Section 5.2 [p.98]) and kNN (Section 5.3 [p.103]) experimental evaluation.

In Figure 5.1 we report the average percentage of involved nodes for increasing M performing 100 random Range queries with $r = 1000$ (which reported an average of 189 results per Range queries) for different number of nodes (varying from 2 to 1024). The results show that for any given number of pivots there is an optimum M in terms of percentage of involved nodes. In particular, for small number of nodes, best results are achieved with small M .

This behavior is due to two distinct phenomena. As the di-

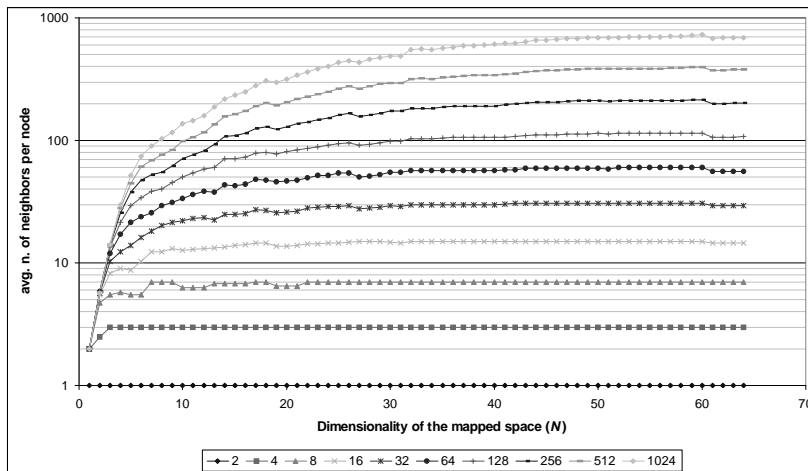


Figure 5.2: Average number of neighbors per node for increasing M and various number of nodes

dimensionality of the mapped space grows, the percentage of the mapped space occupied by the Range queries decreases. On the other side, as the M grows, the average side lengths of the zones assigned to the nodes, increases and the percentage of involved nodes is less correlated to the percentage of the mapped space occupied by the Range queries. Moreover, it is significantly greater than the occupied space percentage.

A well known effect of increasing the dimensionality of a CAN, is the increasing in number of neighbors per node. In Figure 5.2 we report the average number of neighbors per node we obtained for the VEC dataset considering different number of nodes and for growing M . Obviously, if $M = 1$ the number of neighbors is always 2 independently from the number of nodes which are forming the network. However, increasing M the number of neighbors per node become highly dependent from the number of nodes, and for $M = 64$ almost all the nodes are neighbors of all the other ones.

As we can see in Figure 5.3, which reports the same results in another form, if we want MCAN to be scalable in terms of number of neighbors per node, good choice of M are between 1 and 4. On

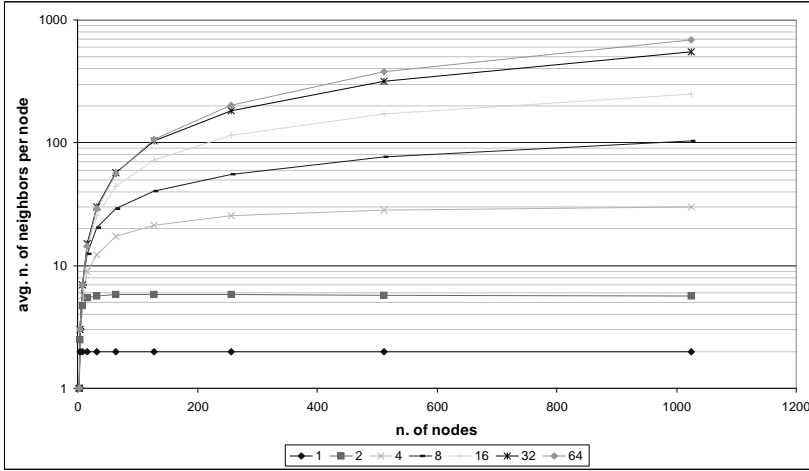


Figure 5.3: Average number of neighbors per node for different M

the other side we saw in Figure 5.1 that, for more than 100 nodes, the optimum M in terms of percentage of involved nodes is above this value.

Thus choosing M in MCAN is a trade-off between number of neighbors per node, which is directly correlated to the average routing path length (see Section 3.2 [p.67]), and the average percentage of involved nodes. In the rest of the thesis we will only use MCAN up to $M = 5$. In most of the cases we will use $M = 3$.

5.2 Range query

In this section we report an evaluation of the Range query algorithm described in Section 4.8 [p.86]. These results were reported in [65].

The metric data-sets used are: 100,000 of 45-dimensional vectors of color features extracted from images (a subset of the VEC dataset); 100,000 Czech sentences of length between 20 and 300 characters. Vectors are compared by the *quadratic-form distance* [158] while for sentences we use the Edit distance (see Subsection 1.3.1 [p.9]).

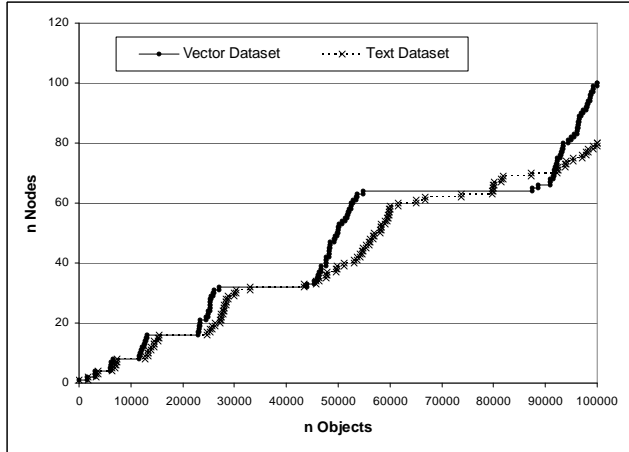


Figure 5.4: Number of nodes for increasing data-set size.

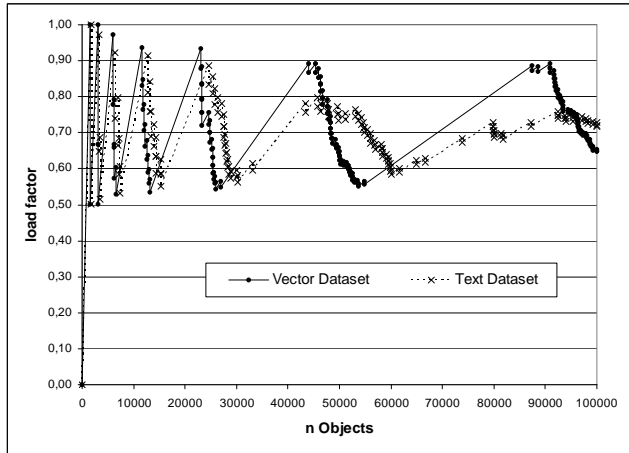


Figure 5.5: Average number of objects per node for the MCAN³ and for increasing data-set size.

We analyze the behavior of the structure in different dimensional spaces: from 1-d (i.e. involving one pivot), to 5-d space (i.e. involving five pivots). As already explained, we use the pivots also to reduce the number of distance computations during the query evaluation on individual nodes. However, independently of the number of dimensions M used by MCAN^M , we always generate 10 pivots in the experiments and we use the first M pivots for creating the MCAN^M zones. Moreover, all 10 pivots are used for filtering during a range query execution internally in nodes.

To study the scalability of the system, we fix the storage space available for each node and then, starting from a single server, we add objects into the system. When a server reaches its storage space limit, it splits. The limit was chosen in a way that after all the 100,000 objects have been inserted, the MCAN^M is composed of around 100 nodes. The node from which an insert operation or a range query starts is randomly selected. Moreover, in order to study the scalability of the system we perform a range query operations every 5,000 insertions.

In Figure 5.4, we report the number of nodes in the system as the data-set grows, for the MCAN^3 case (the other cases are very similar). Note from these experiments that, the number of nodes exhibit a stepwise behavior. This is due to the fact that the objects are randomly ordered, therefore the nodes are filled uniformly and then they tend to split at the same time. This is particularly evident for the vector data-set, where the objects have a fixed size, while the size of objects of the text data-set (strings) is variable.

In Figure 5.5, we report the average load factor for both the data-sets. We define the load factor as the total number of objects stored into the MCAN structure divided by the capacity of storage available on all nodes. As can be seen in the figure, the values are always between 0.5 and 1. This is always guaranteed, because when a node is split, half of the objects are migrated to the new node; therefore the node occupation cannot be less than 50%.

For the performance evaluation of range queries, we selected 100 random objects from the data-set and for each of them we performed 8 different range queries every 5,000 insert operations.

We do not report the average result set size for the different query radii, since they are linear to the data-set size. However, the heaviest range queries return around 3% of the objects for both vector and text data-sets. Note that, these results are independent from the type of access structure but depend on specific characteristics of the given data-sets.

In Figure 5.6 and Figure 5.7 we report the average percentage of nodes involved during a range query operation for different radii as the data-set size grows. Observe that the bigger is the radius of the range query, the more the nodes involved in the query evaluation are. In a naive distributed system we could randomly distribute the objects among the nodes but in this case we would always involve all the nodes even for small radii.

For simple operations like the exact match, the standard CAN has been proved to be scalable. MCAN extends CAN by allowing similarity operations over generic metric space data-sets. In this scenario, we must be able to perform more complex operations such as similarity range queries. To preserve scalability also for such operations, we need more nodes as the complexity of the query grows. This aspect is evident in the plots of Figure 5.6 and Figure 5.7, where the percentage of nodes involved for a small radius is smaller than the ones we obtain for greater radii. Note that, for a given range query, the percentage of nodes involved is almost constant. In fact, for a given range query the number of results is linearly dependent on the number of objects in the data-set and then the number of nodes involved is proportional to the number of results.

To study the complexity of the range queries, we use the number of distance computations. However, for the case of the edit distance (i.e. the Czech-sentences data-set) we must consider the fact that the complexity of a single distance computation is not constant but it is proportional to the string lengths. In this case we decided to use the *equivalent complexity of the edit distance* defined as $L(a)L(b)/\mu(L)^2$, where a , b are two strings evaluated with the edit distance, $L(\cdot)$ is the length of the string, and $\mu(L)$ is the average length of the strings of the data-set.

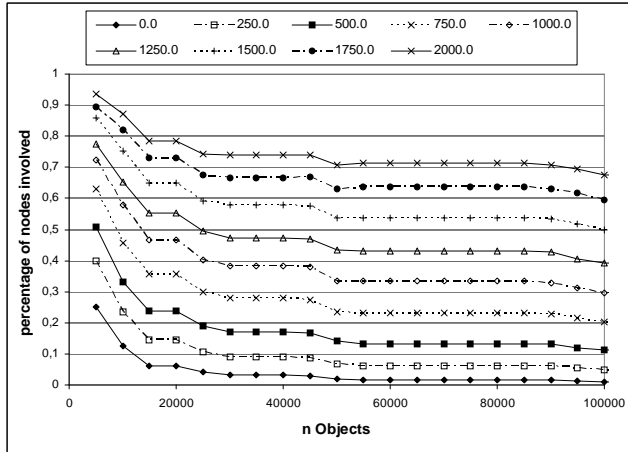


Figure 5.6: Percentage of nodes involved in the Range query as function of the data-set size for different radii (vector data-set).

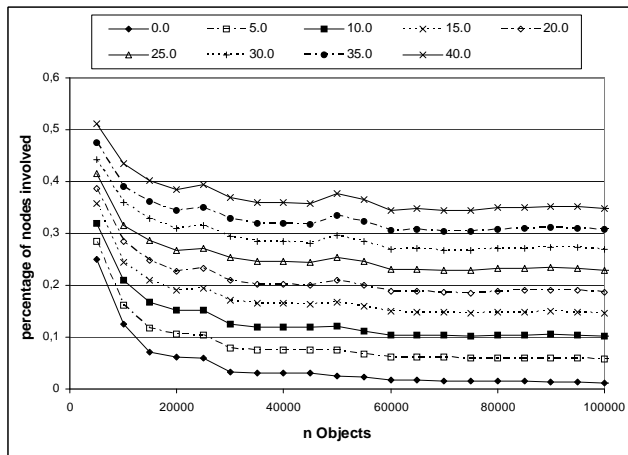


Figure 5.7: Percentage of nodes involved in the Range query as function of the data-set size for different radii (text data-set).

In the reported experiments, we used three pivots for building MCAN structure (i.e. MCAN³) and 16 pivots for filtering.

In Figures 5.8 and 5.9, we report the average complexity of the range query operations as function of the number of equivalent distance computations of the most stressed node. This quantity measures in a way the *intraquery parallelism* as the parallel response time of a range query, if we neglect the message latency. In fact, the requesting node will have to wait the answer of all the involved nodes and then the response time of the query will be proportional to the number of distance computations of the most stressed node. Obviously this quantity is upper bounded by the capacity of the nodes of the MCAN. However, our experiments show that for most of the ranges, the intraquery parallelism remains quite lower than this upper bound, which, for example, in the case of the vector data-set is 1,542.

5.3 Nearest Neighbor query

In this section we report an evaluation of the kNN algorithms described in Section 4.9 [p.88]. These results were reported in [66].

For these experiments, the systems consisted of up to about 300 active peers (depending on the dataset). The peers had storage capacity of 5,000 objects. The implementations built up overlay structures over a high-speed LAN communicating via the TCP and UDP protocols.

It is important to observe that in order to evaluate the scalability performance of MCAN, in this experimental evaluation we maintain the list of available inactive peers and employ them during the split of an overloading peer. We are aware of the fact that maintaining the list of the inactive peers is quite unusual in a real P2P scenario; however, this approach was adopted just to study the scalability of MCAN with respect to a growing dataset, assuming that the greater the dataset the more peers are employed. The objective was to demonstrate that keeping the average number of objects per peer limited as the dataset grows, the response time of

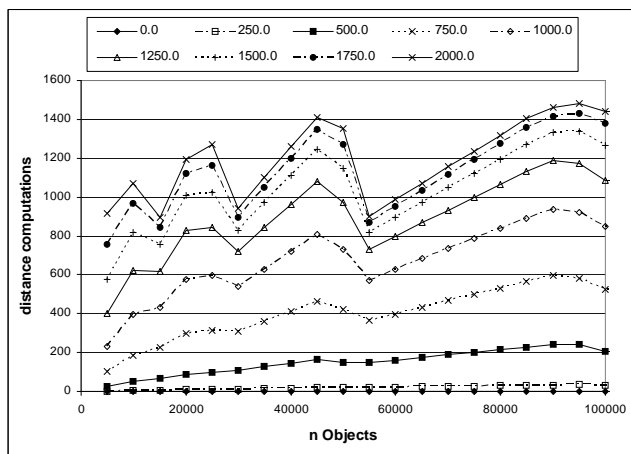


Figure 5.8: Average number of distances evaluated by the most stressed node for each query and for different query range (vector data-set).

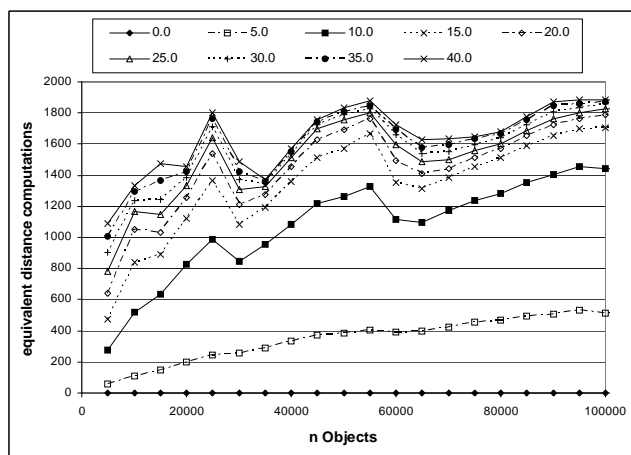


Figure 5.9: Average number of equivalent distances evaluated by the most stressed node for each query and for different query range (text data-set).

the system remains bounded, i.e. the structure scales well. These experiments were performed by inserting objects in the network in a random order, resulting in a growing number of peers due to the splitting rule described in Section Section 4.6 [p.84].

We selected the following significantly different real-life datasets to conduct the experiments on:

VEC 45-dimensional *vectors* of extracted color image features. The similarity of the vectors was measured by a *quadratic-form distance* [158]. The distribution of the dataset is quite uniform and such a high-dimensional data space is extremely sparse.

TTL titles and subtitles of Czech books and periodicals collected from several academic libraries. These *strings* were of lengths from 3 to 200 characters and are compared by the *edit distance* [102] on the level of individual characters. The distance distribution of this dataset is skewed.

DNA protein symbol *sequences* of length sixteen. The sequences were compared by a *weighted edit distance* according to the Needleman-Wunsch algorithm [133]. This distance function has quite a limited domain of possible values — the returned values are integers between 0 and 100.

Please observe that none of these datasets can be efficiently indexed and searched by a standard vector data structure.

All the presented performance characteristics of query processing have been taken as an average over 100 queries by randomly choosing query objects not belonging to the dataset.

It is important to remark that, in a real scenario as the one we are evaluating, the calculation of the distance function d has typically a high computational cost. Therefore, the main objective of a metric-based data structures is to reduce the number of distance computations at query time. The number of distance computations is typically considered an indicator of the structure efficiency. In practice, we assume that the costs of other operations are negligible compared to the distance evaluation time.

Concerning the distributed environment, we use the following two characteristics to measure the computational costs of a query:

- *total distance computations* — the sum of the number of distance computations on all employed peers,
- *parallel distance computations* — the maximal number of distance computations performed in a sequential manner during the parallel query processing.

Another indicator that we monitored is the *percentage of peers* (with respect to the total number) that were involved by the query processing and the *number of candidate results*.

In order to better interpret the performance figures of the three kNN algorithms presented above, we compare the results of the experiments with an ideal kNN algorithm, designated RQ, which is equivalent to a single range query. RQ works as follows: once we have obtained the result set of the kNN (evaluated using one of the three algorithms), we run a range query with radius $d(x_k, q)$. The performance figures of RQ can be considered as the lower bounds (optimal) for the other kNN algorithms.

5.3.1 Number of peers involved in query execution

Figure 5.10 shows the average percentage of involved peers during the evaluation of kNN for increasing values of k and for the entire dataset. The performance figures as function of k are only reported for the VEC dataset (1 million objects and 260 peers), since the results of the other two datasets are very similar.

From these experiments, we can see that SE is not only the best algorithm in terms of number of involved peers but it is also optimum. In fact, its results are the same as those we obtained with RQ. PE involves much more peers and MME is not far from PE. Note also that, the number of peers grows almost linearly with k . Moreover, we can observe that for bigger k MME tends to be slightly better than PE. In fact, the more are the peers involved,

the more is the relevance of intermediate results updated during the forwarding of the operation.

Figures 5.11, 5.12, and 5.13 show the average percentage of involved peers as the dataset grows. For all algorithms, the percentage of involved peers decreases with the number of objects of the dataset. This can be explained by the fact that in general as the dataset grows the distance of the k -th object to the query q decreases. Furthermore, as the number of peers increases, the average volume of zones maintained by the peers decreases.

Regarding the differences between the three datasets' results, we can see that the DNA dataset is much more difficult to index than the VEC one. The most important reason is that the DNA metric function has a very limited number of discrete distance values. The TTL dataset is in the middle but not far from the DNA dataset. In fact, the TTL and DNA distance functions are not much different even if the objects are, in their meaning, completely different.

5.3.2 Total number of distance computations

In Figure 5.14, we report the total number of distance computations during the kNN operation for the entire dataset and various k . Even though not optimum, SE is very near to the results obtained with RQ. In fact, it was also the algorithm that involved less peers. The price of this good result is the serialization of the operation which we will study more in details later on. Regarding MME, we can see that performing kNN computations before forwarding the query significantly reduces the total number of distance computations.

Figures 5.15, 5.16, and 5.17 show the total number of distance computations as the dataset size grows. Note that, in all the algorithms the total number of distance computations grows when we increase the dataset size. However, the most important property that we should expect from a P2P data structure such as the MCAN is its scalability of the search operations, which is achieved through parallelism.

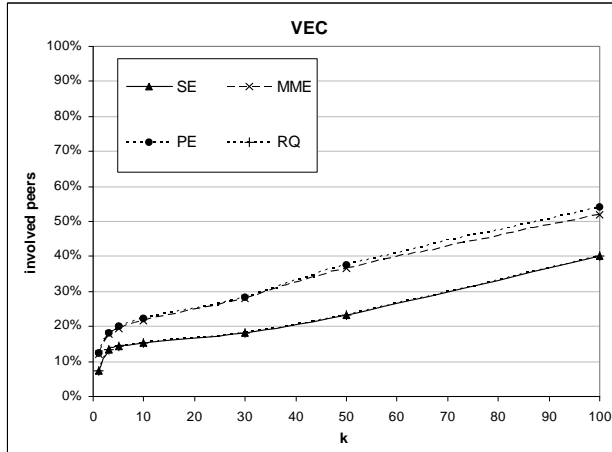


Figure 5.10: Percentage of involved peers for various k for VEC dataset.

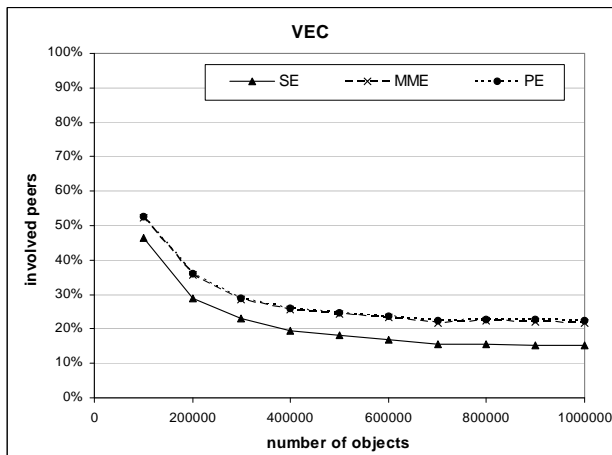


Figure 5.11: Percentage of involved peers for growing VEC dataset ($k = 10$).

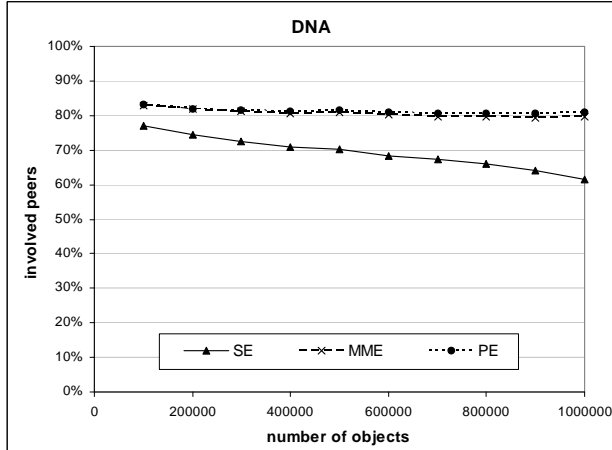


Figure 5.12: Percentage of involved peers for growing DNA dataset ($k = 10$).

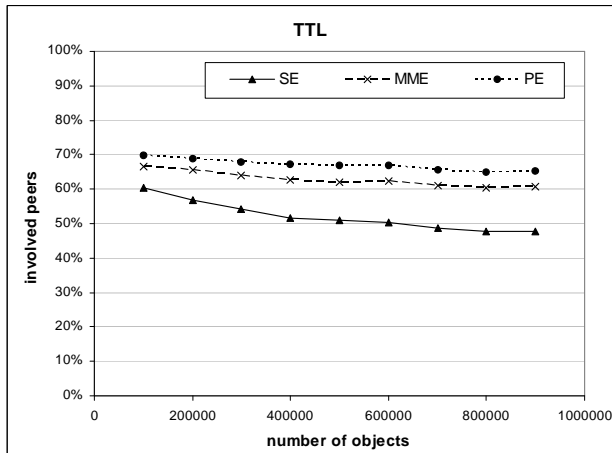


Figure 5.13: Percentage of involved peers for growing TTL dataset ($k = 10$).

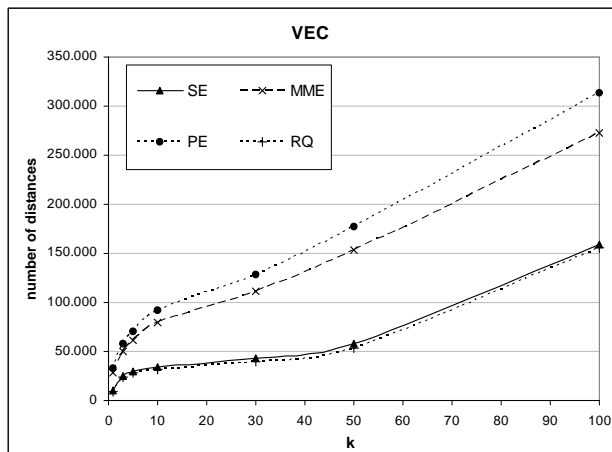


Figure 5.14: Total number of distance computations for various k for VEC dataset.

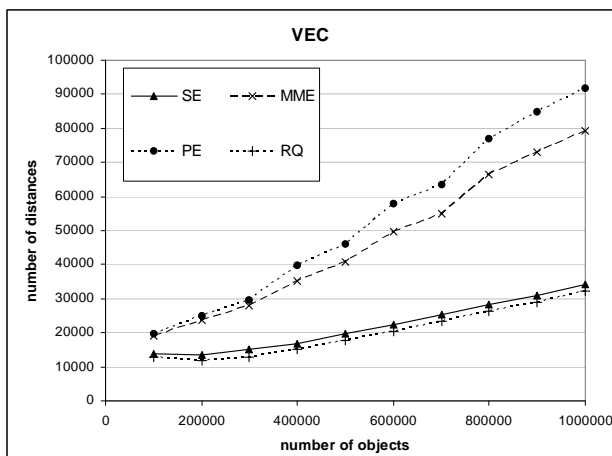


Figure 5.15: Total number of distance computations for growing VEC dataset ($K=10$).

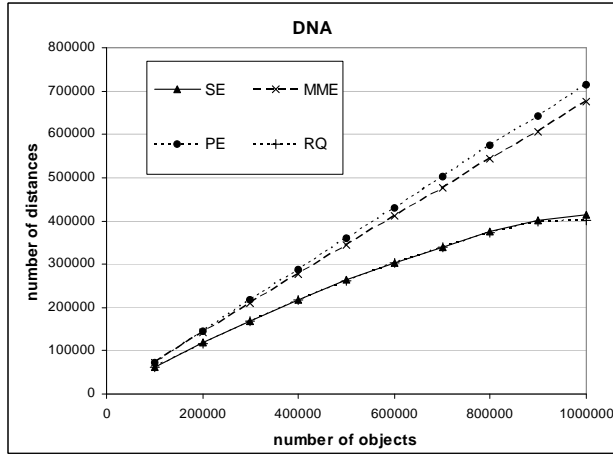


Figure 5.16: Total number of distance computations for growing DNA dataset ($K=10$).

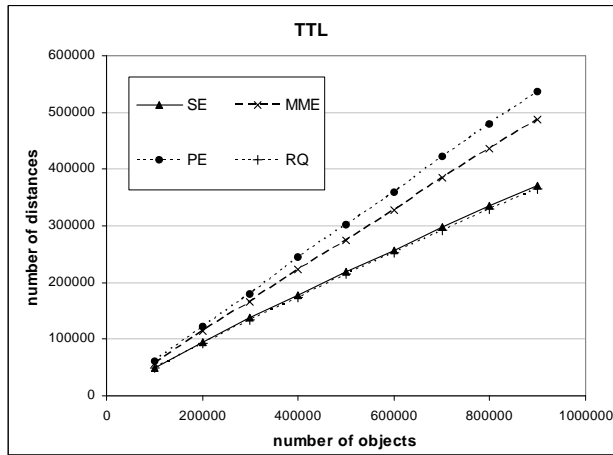


Figure 5.17: Total number of distance computations for growing TTL dataset ($K=10$).

5.3.3 Parallel cost of kNN

In Figure 5.18, we report the parallel distance computations for increasing values of k . As explained above, this performance figure can be considered as the parallel cost of the operation. From this experiment, it becomes clear what is the price the SE algorithm has to pay to obtain better results in terms of percentage of involved peers and total number of distance computations — the cost of the operation grows quickly with k . Furthermore, from Figures 5.19, 5.20, and 5.21 we can see that the parallel cost of SE grows with the dataset size, which means that the algorithm does not scale well. On the contrary, PE scales well and it is very near to the optimum RQ. Note that, the parallel cost of RQ does not grow with the dataset size, because as the number of objects increases the corresponding range radius $d(x_k, q)$ becomes smaller. Finally, MME, which gave better results than PE in terms of the number of involved peers and total number of distance computations, does not scale as well as the PE, but it is not very far from it.

Considering the parallel cost as the response time of the network it must be noticed that, because the results are sent by each involved peer directly to the requester, there are some preliminarily results before the kNN operation is completed. Regarding SE and MME, the order in which the peers are visited guarantees that good results will be available to the requester before the end of the operation. However, in the PE algorithm the results are supposing to arrive sooner and almost at the same time because of the parallelism (except for the $\mu(\hat{q})$ peer), although the use of preliminarily results by the requester can make MME and SE more appealing in some scenarios.

5.3.4 Candidate results

In all the algorithms, as the kNN evaluation proceeds, the peers send the partial results of their local kNN evaluation to the requesting peer $\mu(\hat{q})$ (which is the peer which started the kNN operation). We call these partial results *candidate results*.

Since the user needs an ordered list of k results, $\mu(\hat{q})$ performs

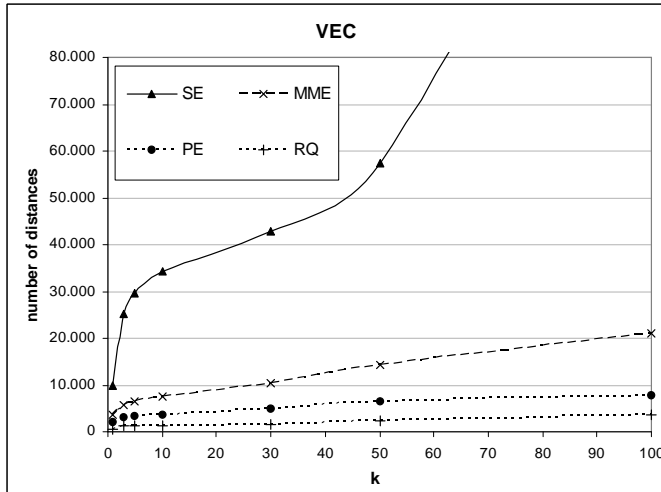


Figure 5.18: Parallel distance computations for various k for VEC dataset.

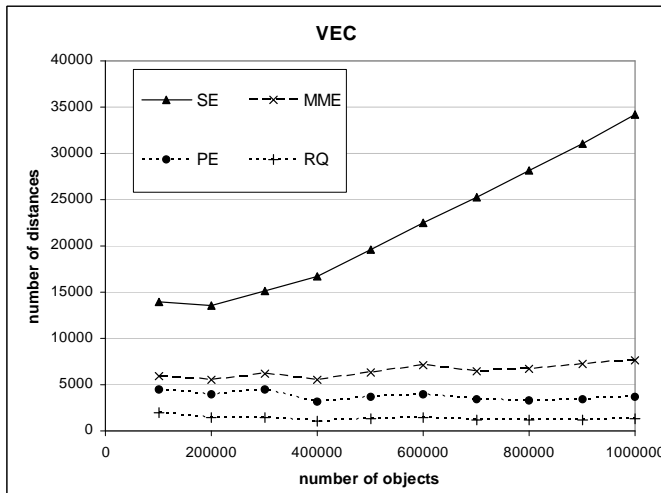


Figure 5.19: Parallel distance computations for growing VEC dataset ($K=10$).

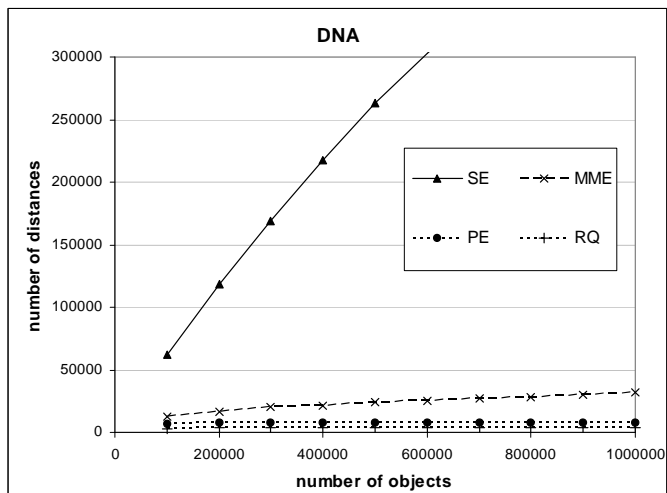


Figure 5.20: Parallel distance computations for growing DNA dataset ($K=10$).

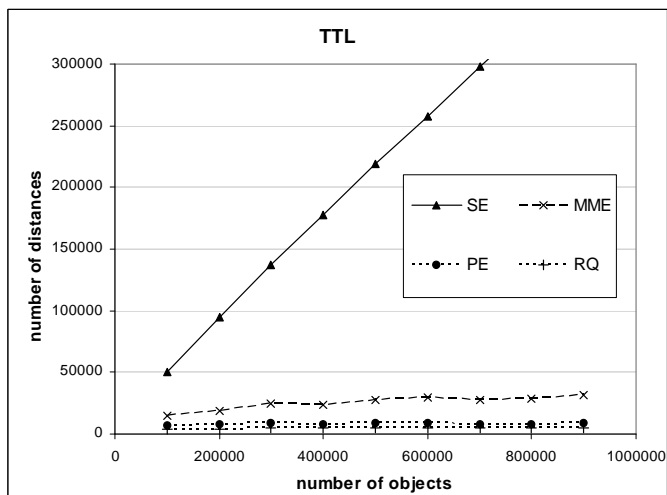
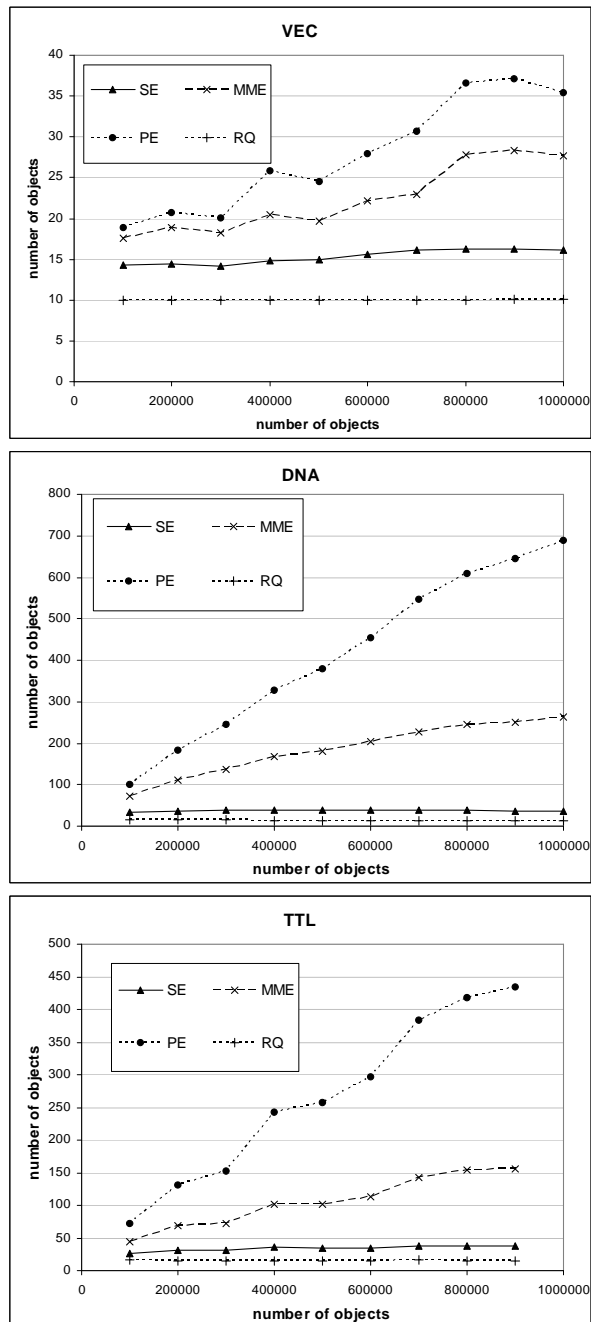


Figure 5.21: Parallel distance computations for growing TTL dataset ($K=10$).

also the task of merging, sorting, and pruning, if necessary, the results beyond the k -th received from the peers involved in the kNN . In Figure 5.22, we report the number of candidate results received for the three algorithms as the dataset grows. The SE algorithm is near the optimum. In fact, it is very near to RQ which, by definition, exactly retrieves k results (it can retrieve more results just in case there are more objects at the same distance of the k -th result). The PE algorithm is the worst and it does not scale well (except for the DNA and TTL datasets). On the contrary, MME is between the other two algorithms and its behavior seems growing sub-linearly.

Notice that, generally, the metric distance evaluation in the metric space is very expensive and the cost of the operations performed by the requester is then negligible. In fact, it does not have to evaluate any distance (we assumed that the distance between the objects and the query are sent together with the results). With these experiments we just wanted to show that there could be problems of scalability in terms of candidate results for the PE algorithm.

Figure 5.22: Average n. of candidate results ($K=10$).

Chapter 6

MCAN Comparison with other structures

In this Chapter, we report on implementations of the VPT*, GHT*, MCAN and M-Chord systems over the same infrastructure of peer computers. We have conducted numerous experiments on three different datasets and present our most telling findings. We focus on scalability with respect to the size of the query, the size of the dataset, and the number of queries executed simultaneously. The results reported in this paper have been presented at the INFOSCALE '06 conference [22] and have been also selected for publication on Future Generation Computer Systems.

We provide a comparison of these four approaches through the results of our extensive experiments. For each data structure, the tests have been conducted on the same datasets and in the same test environment. Moreover, all the structures have been implemented over the very same infrastructure sharing a lower-level code. Consequently, we consider the results of these experiments sufficiently comparable.

When designing the experiments, we focused on particular aspects of the scalability of the systems. Namely, we studied scalability with respect to the query selectivity, with respect to the size of the indexed dataset, and in consideration of the number of queries executed concurrently.

6.1 Experiments Settings

All the compared systems are dynamic. Each structure maintains a set of available inactive nodes employing them when splitting overloaded nodes. For the experiments, the systems consisted of up to 300 active nodes. Each of the GHT* and VPT* peers maintained five buckets with capacity of 1,000 objects and the MCAN and M-Chord peers had a storage capacity of 5,000 objects. The implementations built the overlay structures over a high-speed LAN communicating via the TCP and UDP protocols.

We selected the following significantly different real-life datasets to conduct the experiments on (they are the same used in Section 5.3 [p.103]):

VEC 45-dimensional *vectors* of extracted color image features.

The similarity of the vectors was measured by a *quadratic-form distance* [158]. The distribution of the dataset is quite uniform and such a high-dimensional data space is extremely sparse.

TTL titles and subtitles of Czech books and periodicals collected from several academic libraries. These *strings* were of lengths from 3 to 200 characters and are compared by the *edit distance* [102] on the level of individual characters. The distance distribution of this dataset is skewed.

DNA protein symbol *sequences* of length sixteen. The sequences were compared by a *weighted edit distance* according to the Needleman-Wunsch algorithm [133]. This distance function has quite a limited domain of possible values — the returned values are integers between 0 and 100.

Observe that none of these datasets can be efficiently indexed and searched by a standard vector data structure.

If not stated otherwise, the stored data volume is 500,000 objects. When considering the scalability with respect to the growing dataset size, larger datasets consisting of 1,000,000 objects are used (900,000 for TTL). As for other settings specific to particular data

structures, the MCAN uses 4 pivots to build the routing vector space and 40 pivots for filtering. The M-Chord uses 40 pivots as well. The GHT* and VPT* structures use variable numbers of filtering pivots according to the depth of the AST tree (see Subsection 2.5.1 [p.53]).

All of these performance characteristics of query processing have been obtained as an average over 100 queries with randomly chosen query objects.

6.2 Measurements

In real applications as well as in the described datasets, evaluating the distance function d typically makes high computational demands. Therefore, the objective of metric-based data structures is to decrease the number of distance computations at query time. This value is typically considered an indicator of structure efficiency. The CPU costs of other operations (and often I/O costs as well) are practically negligible compared to the distance evaluation time.

Concerning the distributed environment, we use the following two characteristics to measure the computational costs of query processing:

- *total distance computations* — the sum of the number of the distance function evaluations on all involved peers,
- *parallel distance computations* — the maximal number of distance evaluations performed in a sequential manner during query processing.

Note that the total number corresponds to costs on a centralized version of the specific structure. The communication costs of a query evaluation are measured by the following indicators:

- *total number of messages* — the number of all messages (requests and responses) sent during a particular query processing,

- *maximal hop count* — the maximal number of messages sent in a serial way in order to complete the query.

Since the technical resources used for testing were not dedicated but opened for public use, the actual query response times were fluctuating and we cannot report them precisely. However, we have usually observed that one range query evaluation took less than one second for small radii and approximately two seconds for the big ones regardless of the dataset size. The parallel distance computations together with the maximal hop count can be used as a fair response time estimation. Another indicator that we monitored is the *percentage of nodes* that were involved in processing a particular query.

6.3 Scalability with Respect to the Size of the Query

In the first set of experiments, we have focused on the systems' scalability with respect to the size of the processed query. Namely, we let the structures handle a set of $R(q, r)$ queries with growing radii r . The size of the stored data was 500,000 objects. The average load ratio of nodes for all the structures was 60–70% resulting in approximately 150 active nodes in each of the systems.

We present results of these experiments for all the three datasets. All graphs in this section represent the dependency of various measurements (vertical axis) on the range query radius r (horizontal axis). The datasets are indicated by titles. For the VEC dataset, we varied the radii r from 200 to 2,000 and for the TTL and DNA datasets from 2 to 20.

In the first group of graphs, shown in Figure 6.1, we report on the relation between the query radius size and the number of objects retrieved. As this depicts, the greater the radius the higher the number of objects satisfying the query. Since we have used the same datasets, query objects and radii, all the structures return the same number of objects. We can see that the number of results grows exponentially with respect to the query radius for all

6.3. SCALABILITY WITH RESPECT TO THE SIZE OF THE QUERY121

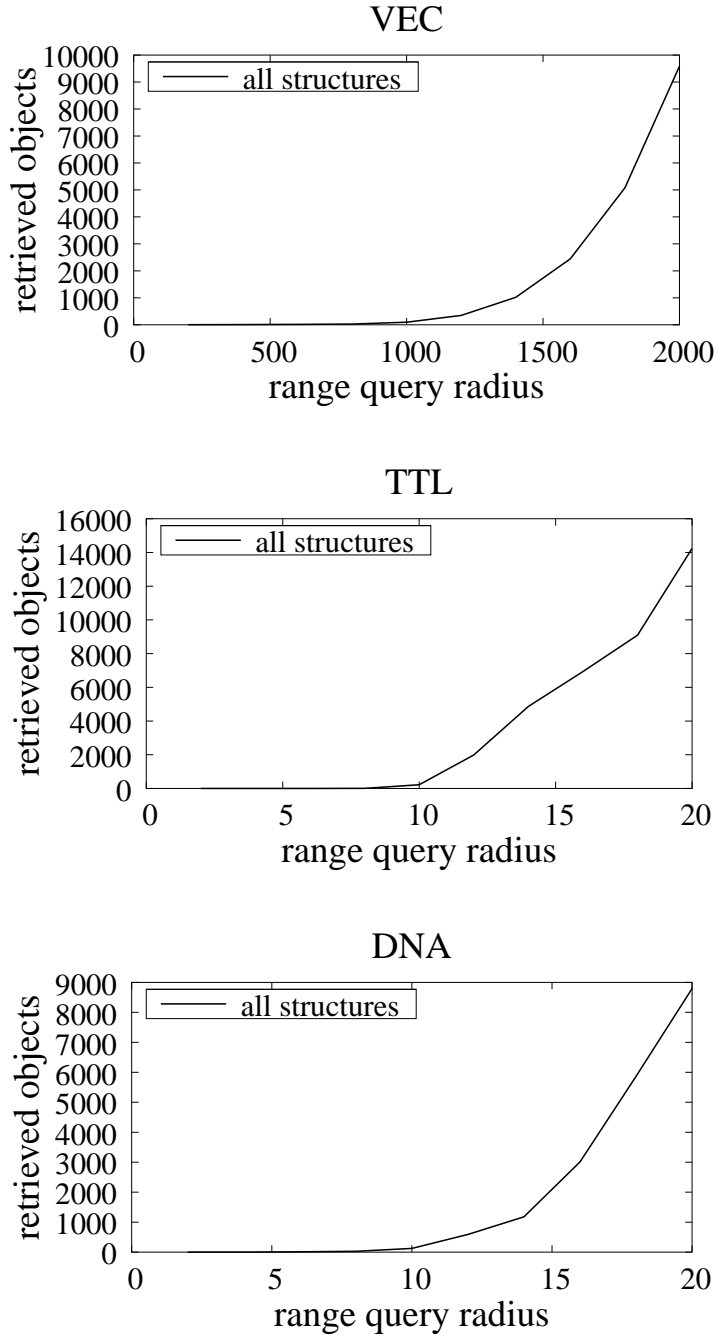


Figure 6.1: Number of retrieved objects

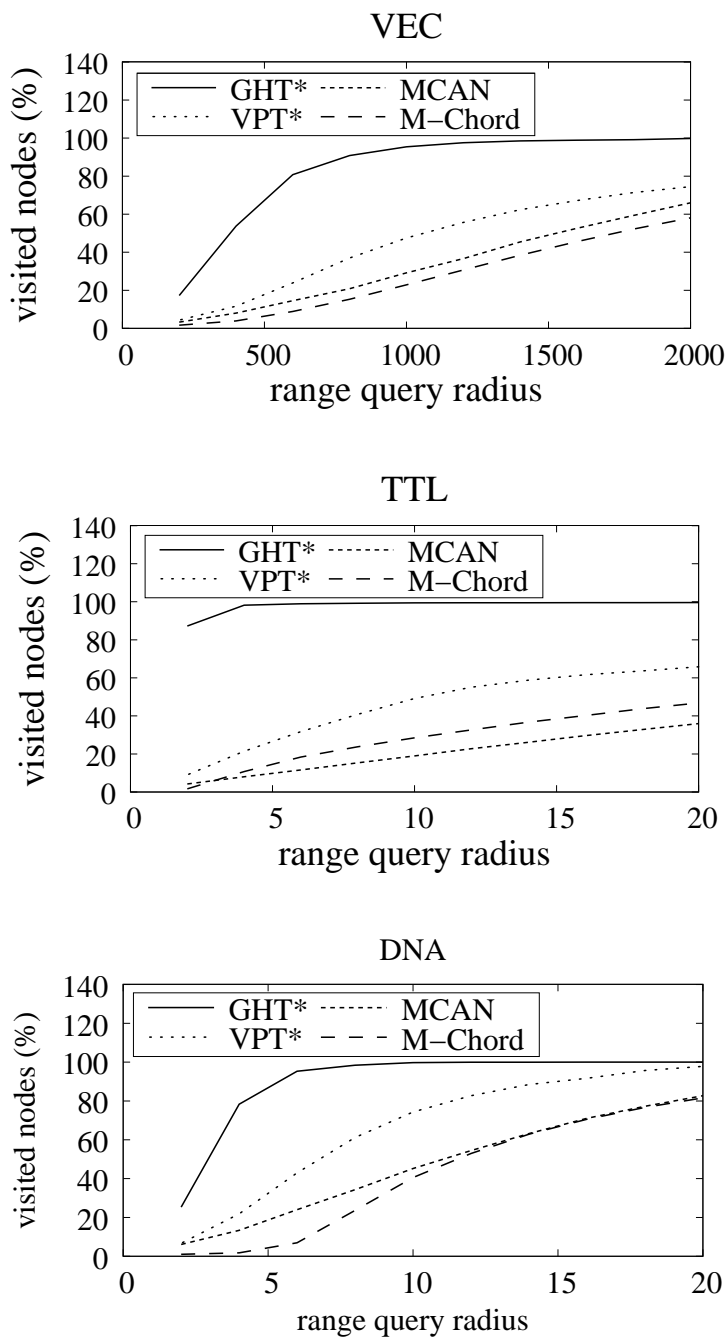


Figure 6.2: Percentage of visited nodes

the datasets. Note that, for example, the greatest radius 2,000 in the VEC dataset selects almost 10,000 objects (2% of the whole database). Obviously, such large radii are not usually practical for applications (e.g. two titles with an edit distance 20 differ considerably), but we provide the results in order to study the behavior of the structures in these cases as well. Smaller radii return reasonable numbers of objects, for instance, radius 6 results in approximately 30 objects in the DNA dataset.

The number of visited nodes is reported in Figure 6.2. More specifically, the graphs show the ratio of the number of nodes that are involved in a particular range query evaluation to the total number of active peers forming the structure. As mentioned earlier, the number of active peers in the network was around 150, thus, value 20% in the graph means that approximately 30 peers were used to complete the query. We can see that the number of employed peers grows practically linearly with the size of the radius. The only exception is the GHT* algorithm, which visits almost all active nodes very soon as the radius grows. This is caused by the fact that the generalized hyperplane partitioning does not guarantee a balanced split, unlike the other three methods. Moreover, because we count all the nodes that evaluate distances as visited, the VPT* and the GHT* algorithms are somewhat handicapped. Recall that they need to compute distances to pivots during the navigation which means that the nodes that only forward the query are also considered visited.

Note that the dataset influences the number of visited nodes. For instance, the DNA metric function has a very limited set of discrete distance values, thus, both the native and transformation methods are not as efficient as for the VEC dataset and more peers have to be accessed. From this point of view, the M-Chord structure performs best for the VEC dataset and also for smaller radii in the DNA dataset, but it is outperformed by the MCAN algorithm for the TTL dataset.

The next group of experiments, depicted in Figures 6.3 and 6.4, shows the computational costs with respect to the query radius. We provide a pair of graphs for each dataset. The graphs reported on

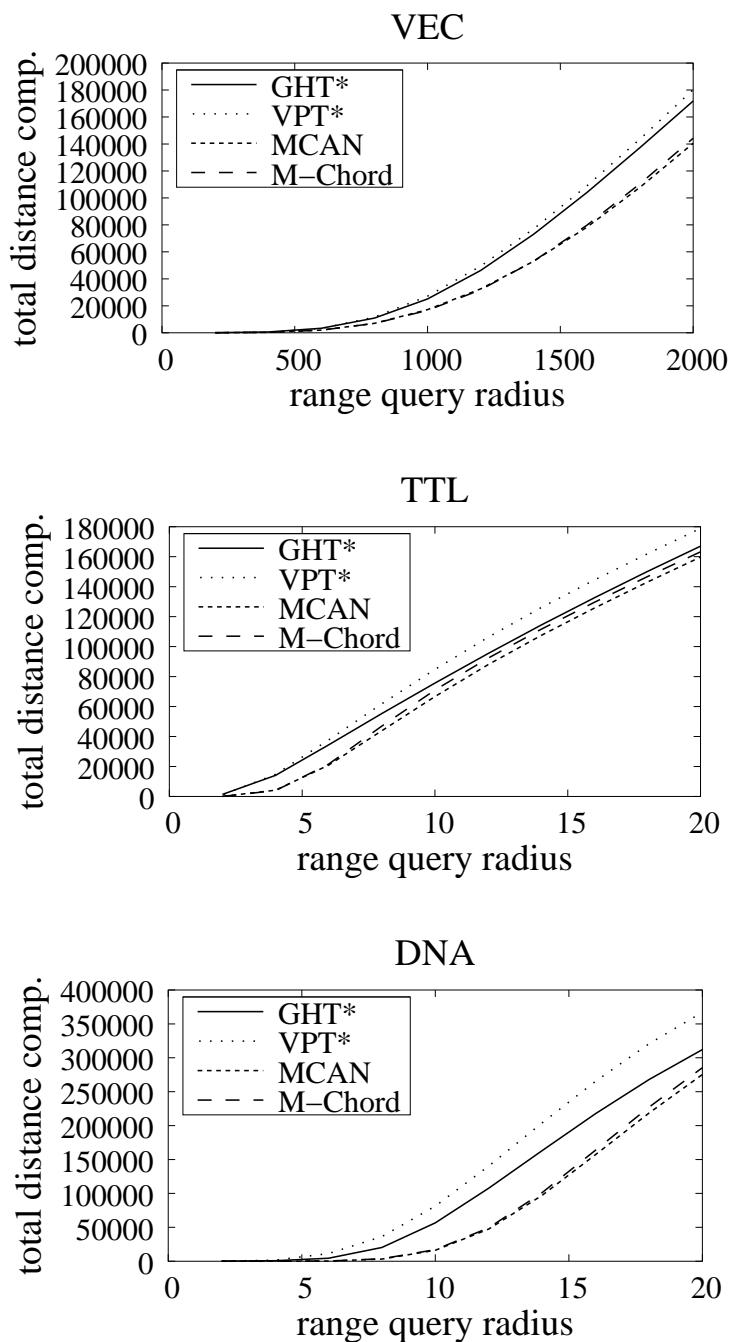


Figure 6.3: The total number of distance computations

6.3. SCALABILITY WITH RESPECT TO THE SIZE OF THE QUERY 125

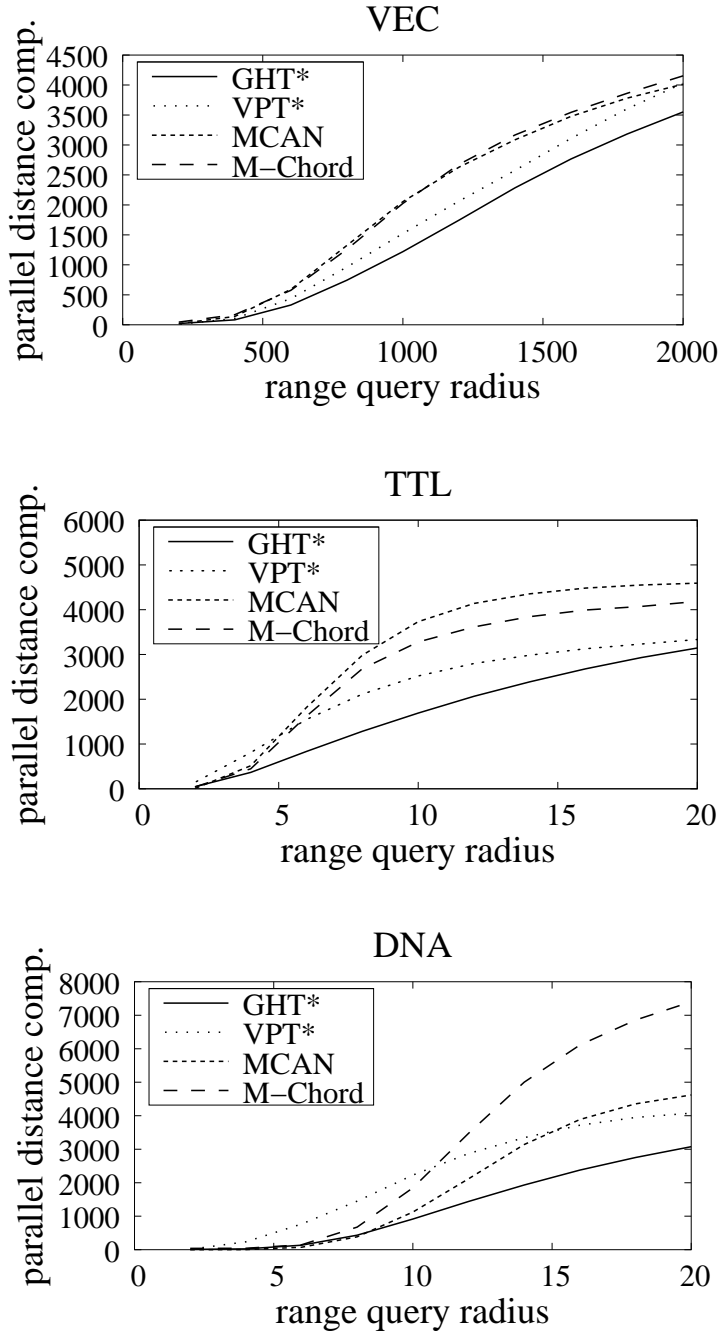


Figure 6.4: The parallel number of distance computations

Figure 6.3 show the total number of distance computations needed to evaluate a range query. This measure can be interpreted as the query costs in centralized index structures. The graphs reported on Figure 6.4 show illustrate the parallel number of distance computations, i.e. the costs of a query in the distributed environment.

Since the distance computations are the most time consuming operations during the evaluation, all the structures employ the pivot filtering criteria to avoid as much distance computations as possible (see Section 4.2 [p.80]). As explained, the number of pivots used for filtering strongly affects its effectiveness, i.e. the more pivots we have the more effective the filtering is and the fewer distances need to be computed. The MCAN and the M-Chord structures use a fixed set of 40 pivots for filtering, as opposed to the GHT* and VPT* which use the pivots in the AST. Thus objects in buckets in lower levels of the AST have more pivots for filtering and vice versa. Also, the GHT* partitioning implies two pivots per inner tree node, but VPT* contains only one pivot, resulting in half the number of pivots used in the GHT*. In particular, the GHT* has used 48 pivots in its longest branch and only 10 in the shortest one, while the VPT* has used maximally 18 and minimally 5 pivots.

Looking at the total numbers of distance computations in Figure 6.3, we can see that the filtering was rather ineffective in the DNA dataset, where the structures have computed the distances for up to twice as many objects than that of the TTL and VEC datasets. The queries with larger radii in the DNA dataset have to access about 60% of the whole database, which would be very slow in a centralized index.

Figure 6.4 illustrates the parallel computational costs of the query processing. We can see that the number of necessary distance computations is significantly reduced, which emerges from the fact that the computational load is divided among the participating peers running in parallel. We can see that the GHT* structure has the best parallel distance computation and seems to be unaffected by the dataset used. However, its lowest parallel cost is counterbalanced by the high percentage of visited nodes (shown in Figure 6.2), which is in fact correlated to the parallel distance

computations cost for all the structures.

Note also that the increase of parallel cost is bounded by the value of 5,000 distance computations — this can be most clearly seen in the TTL dataset. This is a straightforward implication of the fact that every node has only a limited storage capacity, i.e. if a peer holds up to 5,000 objects it cannot evaluate more distance computations between the query and its objects. This seems to be in contradiction with the M-Chord graph for the DNA dataset, for which the following problem has arisen. Due to the small number of possible distance values of the DNA dataset, the M-Chord transformation resulted in the formation of “clusters” of objects mapped onto the same M-Chord key. Those objects had to be kept on one peer only and, thus, the capacity limit of 5,000 objects was exceeded.

The last group of measurements in this section reports on the communication costs measured as the total number of messages sent (Figure 6.5) and as the maximal hop count (Figure 6.6), i.e. the maximal number of messages sent in a serial manner. Since the GHT* and the VPT* count all nodes involved in navigation as visited (as explained earlier), the percentage of visited nodes (Figure 6.2) and the total number of messages (Figure 6.5) are strictly correlated for these structures. The MCAN structure needs the lowest number of messages for small ranges, but as the radius grows, the number of messages increases quickly. This comes from the fact that the MCAN range search algorithm uses multicast to spread the query and, thus, one peer may be contacted with a particular query request several times. However, every peer evaluates each request only once. For the M-Chord structure, we can see that the total cost is considerably high even for small radii, but it grows very slowly as the radius increases. This is caused by the fact that the M-Chord needs to access at least one peer for every M-Chord cluster even for small range queries, see Subsection 2.5.2 [p.57].

The parallel costs, i.e. the maximal hop count, are practically constant for different sizes of the radii across all the structures except for M-Chord in which it grows. This increase is caused by the serial nature of the current algorithm for contacting the

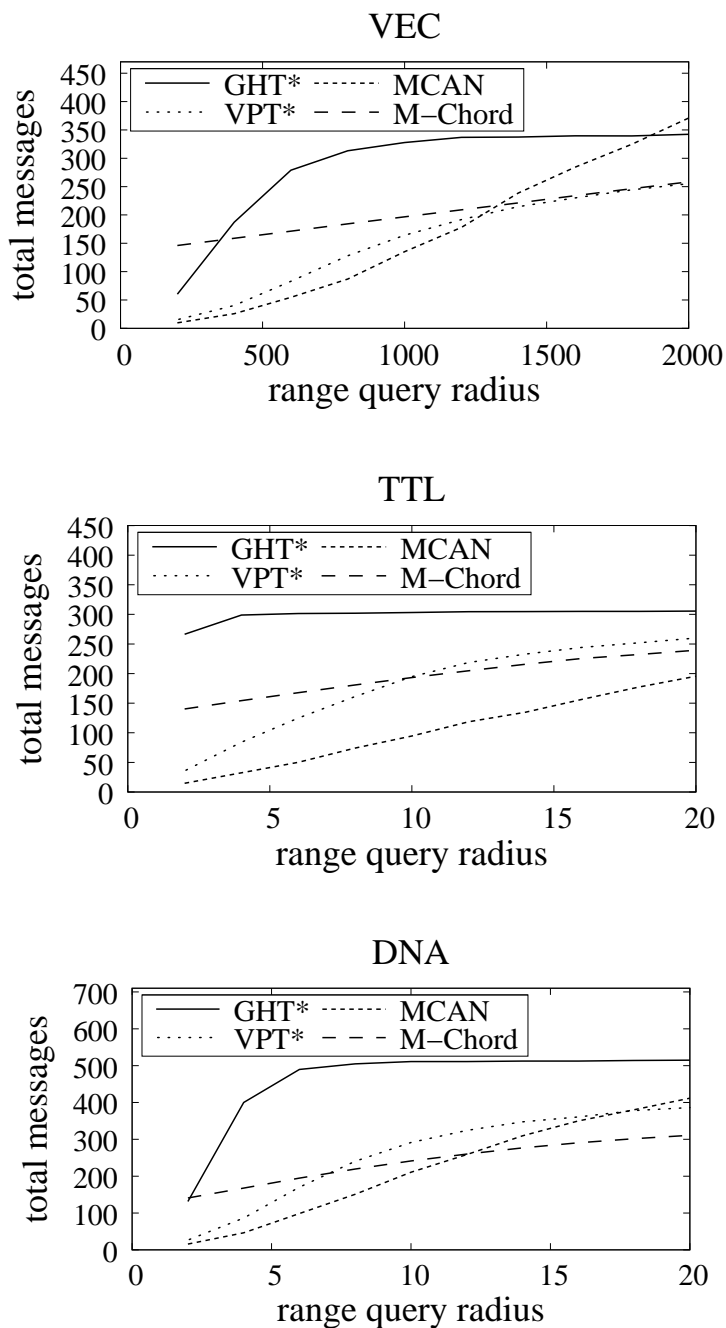


Figure 6.5: The total number of messages

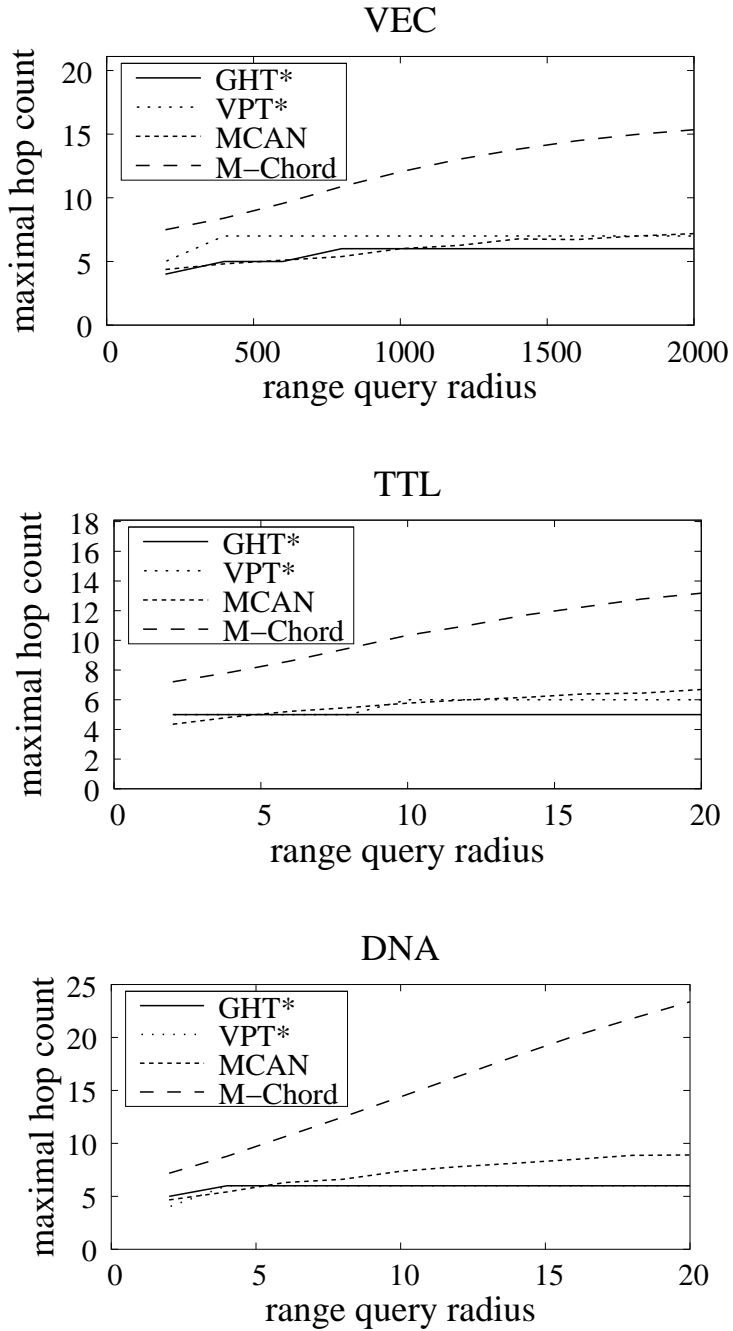


Figure 6.6: The maximal hop count

adjacent peers in particular clusters.

In summary, we can say that all the structures scale well with respect to the size of the radius. In fact, the parallel distance computation costs grow sub-linearly and they are bounded by the capacity limits of the peers. The parallel communication costs remain practically constant for the GHT*, VPT*, and MCAN structures and grows linearly for the M-Chord.

6.4 Scalability with Respect to the Size of Datasets

Let us concern the systems' scalability with respect to the growing volume of data stored in the structures. We have monitored the performance of $R(q, r)$ queries processing on systems storing from 50,000 to 1,000,000 objects. We conducted these experiments for the following radii: 500, 1,000 and 1,500 for the VEC dataset and radii 5, 10 and 15 for the TTL and DNA datasets.

We include graphs for only one dataset for each type of measurement if the other graphs exhibit the same trend. The title of each graph in this section specifies the dataset used and the search radius r .

The number of retrieved objects (see graph for radius 10 and the TTL dataset in Figure 6.7a) grows precisely linearly because the data were inserted to the structures in random order.

Figure 6.7b depicts the percentage of nodes affected by the range query processing. For all the structures but the GHT*, this value decreases because the data space becomes denser and, thus, the nodes cover smaller regions of the space. Therefore, the space covered by the involved nodes comes closer to the exact space portion covered by the query itself. As mentioned in Section 6.3 [p.120], the GHT* partitioning is not balanced, therefore, the query processing is spread over larger number of participating nodes.

Figure 6.8 presents the computational costs in terms of both total and parallel numbers of distance computations. As expected, the total costs (a) increase linearly with the data volume stored.

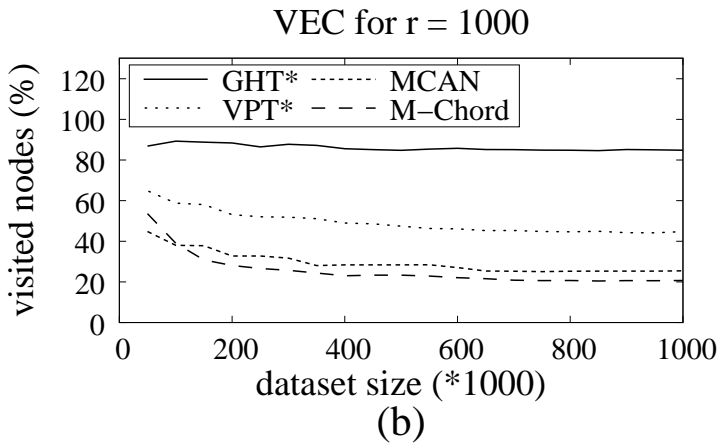
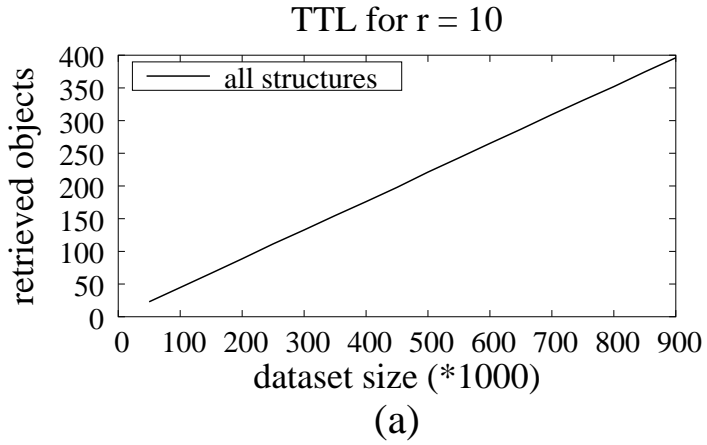


Figure 6.7: Retrieved objects (a) and visited nodes (b) for growing dataset

This well-known trend, which corresponds to the costs of centralized solutions, is the main motivation for designing distributed structures. The graph exhibits practically the same trend for the M-Chord and MCAN structures since they both use a filtering mechanism based on a fixed sets of pivots. The total costs for the GHT* and the VPT* are slightly higher due to smaller sets of filtering pivots.

The parallel number of distance computations (Figure 6.8b) grows very slowly. For instance, the parallel costs for the GHT* increase by 50% while the dataset grows 10 times and the M-Chord exhibits a 10% increment for a doubled dataset size from 500,000 to 1,000,000. The increase is caused by the fact that the nodes involved in the search contain more of the relevant objects while making the data space denser. This corresponds to the observable correlation of this graph and Figure 6.7b — the less nodes the structure involves, the higher the parallel costs it exhibits. The transformation techniques, the MCAN and the M-Chord, concentrate the relevant data on fewer nodes and consequently have higher parallel costs. The noticeable graph fluctuations are caused by quite regular splits of overloaded nodes.

Figure 6.9 presents the same results for DNA dataset. The pivots-based filtering performs less effectively for higher radii (the total costs are quite high) and it is more sensitive to the number of pivots. The distance function is discrete with a small variety of possible values. As mentioned in Section 6.3 [p.120], for this dataset, the M-Chord mapping collisions may result in overloaded nodes that cannot be split. Then, the parallel costs in Figure 6.9b may be over the split limit of 5,000 objects.

Figure 6.10 shows the communication costs in terms of the total number of messages (a) and the maximal hop count (b). The total message costs for the GHT* grow faster because it contacts higher percentages of nodes. The M-Chord graphs indicate that the total message costs grow slowly while the major increase of the messages sending is in a sequential manner which negatively influences the hop count.

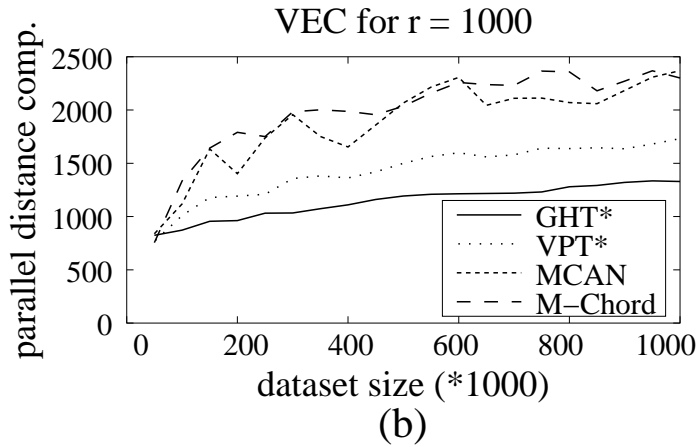
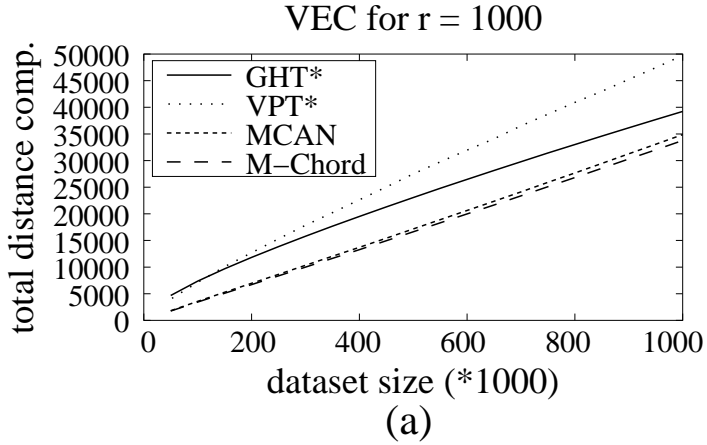


Figure 6.8: The total (a) and parallel (b) computational costs for VEC

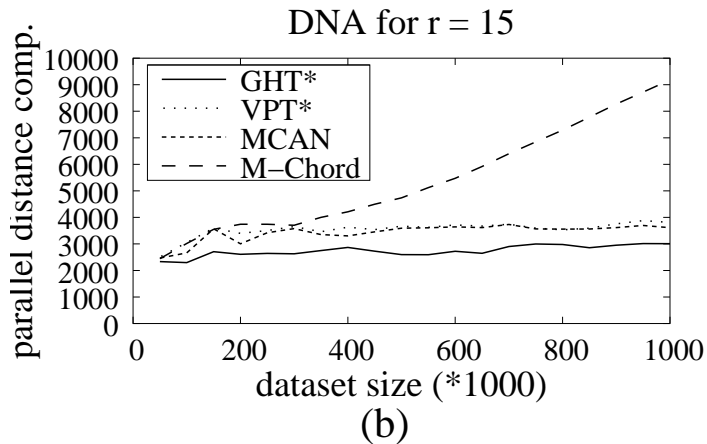
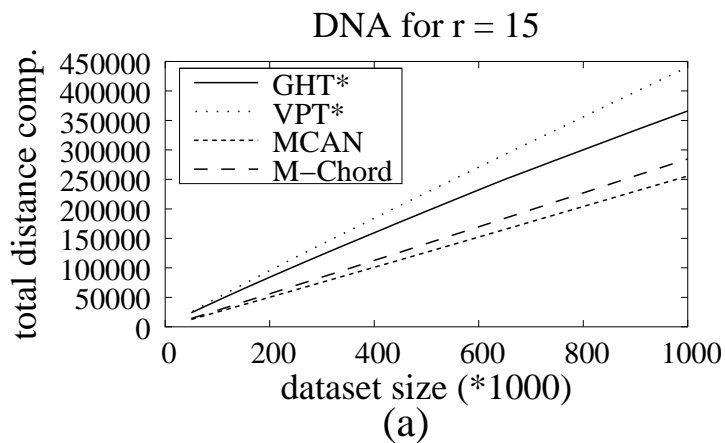


Figure 6.9: The total (a) and parallel (b) computational costs for DNA

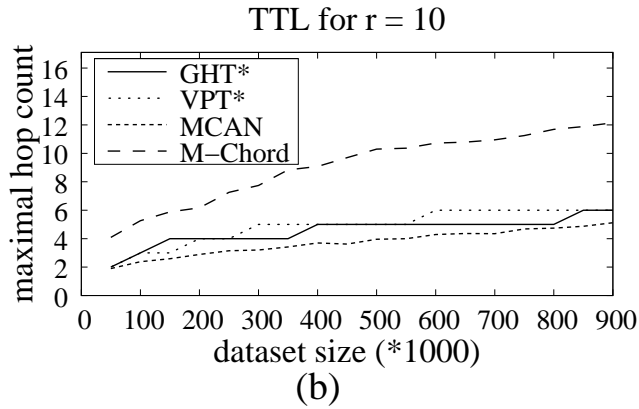
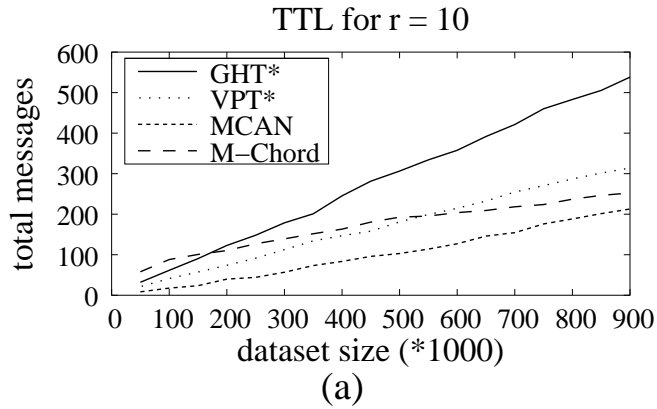


Figure 6.10: The total messages (a) and the maximal hop count (b)

6.5 Number of Simultaneous Queries

In this section, we focus on the scalability of the systems with respect to the number of queries executed simultaneously. In other words, we consider the *interquery* parallelism [196] of the queries processing.

In our experiments, we have simultaneously executed groups of 10 to 100 queries — each from a different node. We have measured the *overall parallel costs* of the set of queries as the maximal number of distance computations performed on a single node of the system. Since the communication time costs are lower than the computational costs, this value can be considered as a characterization of the overall response time. We have run these experiments for all datasets using the same query radii as in Section 6.4.

In order to establish a baseline, we have calculated the *sum of the parallel costs* of the individual queries. The ratio of this value to the *overall parallel costs* characterizes the improvement achieved by the interquery parallelism and we refer to this value as the *interquery improvement ratio*. This value can be also interpreted as the number of queries that can be handled by the systems simultaneously without slowing them down.

Looking at Figure 6.11a, 6.12a and Figure 6.13a, we can see the overall parallel costs for all the datasets and selected radii. The trend of the progress is identical for all the structures and, surprisingly, the actual values are very similar.

Therefore, the difference of the respective interquery improvement ratios, shown in the (b) graphs, is introduced mainly by difference of the single query parallel costs. The M-Chord and the MCAN handle multiple queries slightly better than the VPT* and significantly better than GHT*.

The actual improvement ratio values for specific datasets are strongly influenced by the total number of distance computations spread over the nodes (see Figure 6.3) and, therefore, the improvement is lower for DNA than for VEC.

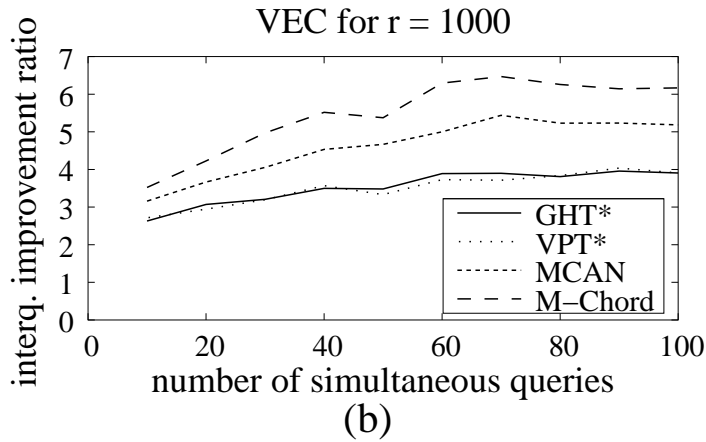
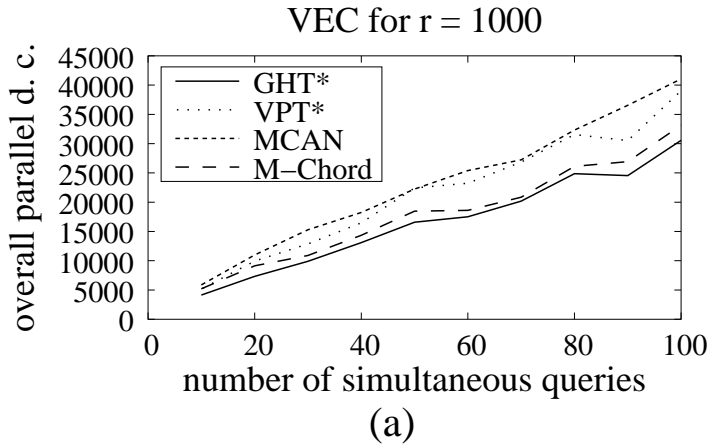


Figure 6.11: The overall parallel costs (a) and interquery improvement ratio (b)

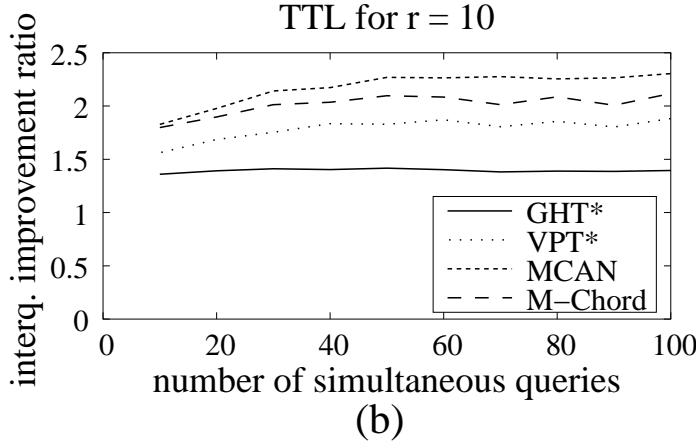
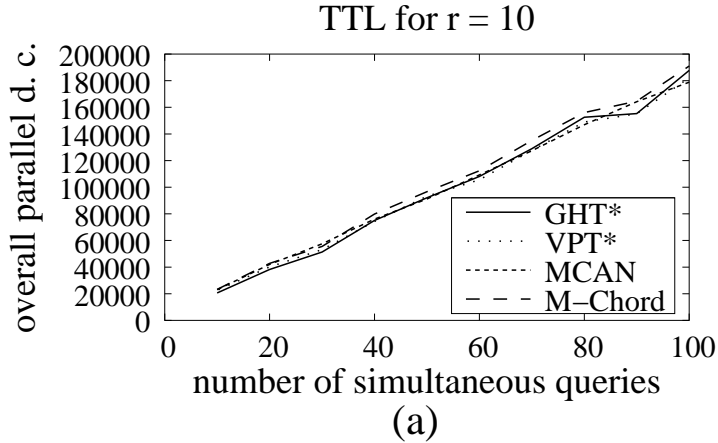


Figure 6.12: The overall parallel costs (a) and interquery improvement ratio (b)

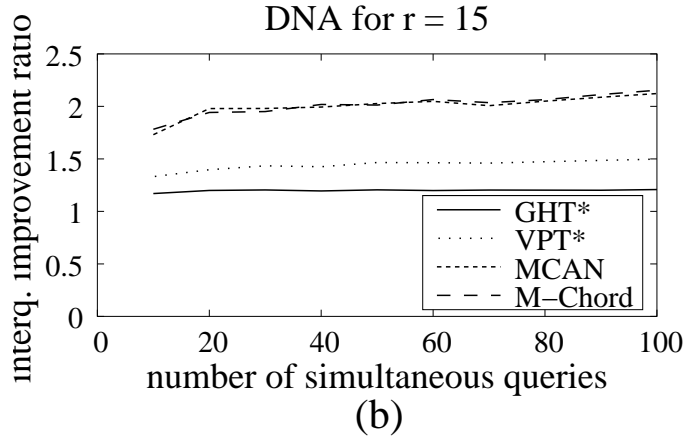
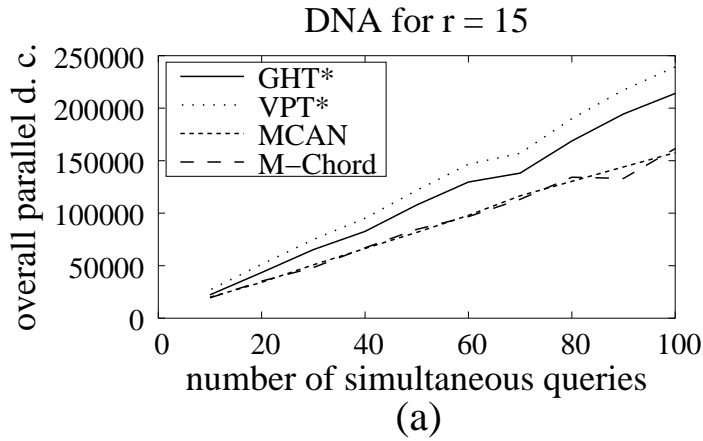


Figure 6.13: The overall parallel costs (a) and interquery improvement ratio (b)

6.6 Comparison Summary

In this paper, we have studied the performance of four different distributed index structures for metric spaces, namely the GHT*, the VPT*, the MCAN and the M-Chord. We have focused on their scalability of executing similarity queries from three different points of view: (1) the changing query radii, (2) the growing volume of data searched, and (3) the accumulating number of concurrent queries. We have conducted a wide range of experiments and reported the most interesting findings in the relevant sections of this paper.

All of the considered approaches have demonstrated a strictly sub-linear scalability in all important aspects of similarity search for complex metric functions. The most essential lessons we have learned from the experiments can be summarized in the following table.

	<i>single query</i>	<i>multiple queries</i>
GHT*	excellent	poor
VPT*	good	satisfactory
MCAN	satisfactory	good
M-Chord	satisfactory	very good

In the table, the *single query* column expresses the power of a corresponding structure to speed up execution of an isolated query. This is especially useful when the probability of concurrent query requests is very low (preferably zero), so only one query is executed at a time and the maximum number of computational resources can be exploited. On the other hand, the *multiple queries* column expresses the ability of our structures to serve several queries simultaneously without degrading the performance by waiting.

We can see that there is no clear winner considering both the single and the multiple query performance evaluation. In general, none of the structures has a poor performance of single query execution, but the GHT* is certainly the most suitable for this purpose. However, it is also the least suitable structure for concurrent query execution — queries in GHT* are almost always processed one after the other. The M-Chord structure has the opposite behavior.

It can serve several queries of different users in parallel with the least performance degradation, but it takes more time to evaluate a single query.

Finally, we would like to emphasize the fact that the transformation-based techniques, i.e. the M-Chord and MCAN, assume having a characteristic subset of the indexed data in advance to choose proper pivots. In our experiments, the assumption was that the distance distribution in the datasets does not change, at least not significantly. If the distribution does change, for example, due to the lack of a characteristic subset during startup, the performance may change. From this point of view, the native organizations are more robust. We plan to systematically investigate this issue hereafter.

In the future, we plan to exploit the pros and cons of the individual approaches revealed by our experiments to design applications with specific querying characteristics. We would also like to use them to develop new search structures combining the best of its predecessors. Future work will also concentrate on performance tuning, which will involve designing structures with respect to the user defined bounds on the query response time.

Chapter 7

Conclusions

In this thesis we investigated the possibility of combining the similarity search indexing techniques with the power of distributed computing infrastructures. More precisely we considered the DHT class of Peer-to-Peer systems for their good performance in performing exact match search through a distributed index.

We proposed MCAN which is a distributed similarity search structure for metric spaces built over the CAN (a well-known DHT). MCAN is based on the concept of choosing pivots to map objects of a generic metric space in a multidimensional vector space whose objects are then distributed using CAN. Since the mapping is contractive, 100% recall for similarity queries processed is guaranteed just contacting a subset of the nodes.

For achieve scalability of the Range and Nearest Neighbor queries with respect to the dataset size, we parallelized their execution. The experiment conducted for growing dataset and proportionally increasing number of nodes, revealed the scalability of the proposed Range query algorithm (see Section 5.2 [p.98] and [65]).

We found that parallelizing the Nearest Neighbor queries is a trade-off between the total and the parallel cost, i.e. between the used resources and the response time. Thus, in Section 4.9 [p.88], we proposed three different algorithms for the nearest neighbors queries execution (see also [66]): a serial one (minimizing the total cost), a parallel one (minimizing the parallel cost) and a mixed

mode (trying to balance parallel and total costs). The results of the experimentations reported in Section 5.3 [p.103] show that the mixed mode is the best choice in general.

The comparison of MCAN Range query with similar existing structure reported in Chapter 6 [p.117] and published in [22] revealed the good performance of the proposed solution.

7.1 Research Directions

An interesting direction of investigation is to generalize the kNN algorithms proposed in Section 5.3 [p.103] by parameterizing the execution parallelism. In the conclusions of [66] we sketched a proposal but we did not implement it yet.

A further interesting direction of investigation is to define an Incremental Nearest Neighbor algorithm which could be less efficient than our kNN algorithm in retrieving the top k results, but more suitable for the execution of complex queries. In fact, state of the art complex queries algorithms (see Subsection 1.2.4 [p.5]), make use of an Incremental Nearest Neighbor algorithm for each feature. Moreover, specific strategies for performing multi features complex similarity queries over multiple MCANs could be studied.

Other possibilities could be investigated. For example, in MCAN each node could have its own centralized similarity index structure instead of the pivoted filtering mechanism described in Section 4.2 [p.80] and used for experiments. We believe that more advanced index structures already presented in the literature or specifically defined for the MCAN nodes could increase the overall performances.

Bibliography

- [1] Gnutella. URL <http://www.gnutella.com>. [Online; accessed 6-November-2006].
- [2] Karl Aberer. P-Grid: A Self-Organizing Access Structure for P2P Information Systems. In *CoopIS '01: Proceedings of the 9th International Conference on Cooperative Information Systems*, pages 179–194, London, UK, 2001. Springer-Verlag. ISBN 3-540-42524-1.
- [3] Karl Aberer. Efficient Search in Unbalanced, Randomized Peer-To-Peer Search Trees. Technical report, 2002.
- [4] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Puceva, and Roman Schmidt. P-Grid: a self-organizing structured P2P system. *SIGMOD Record*, 32(3):29–33, 2003. ISSN 0163-5808.
- [5] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Puceva, Roman Schmidt, and Jie Wu. Advanced Peer-to-Peer Networking: The P-Grid System and its Applications . *PIK journal*, 26(3), 2003.
- [6] Karl Aberer, Anwitaman Datta, and Manfred Hauswirth. P-Grid: Dynamics of Self-Organizing Processes in Structured Peer-to-Peer Systems. In *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*,

- chapter 10, pages 137–153. Springer-Verlag Berlin Heidelberg, 2005.
- [7] Karl Aberer, Fabius Klemm, Toan Luu, Ivana Podnar, and Martin Rajman. Building a peer-to-peer full-text Web search engine with highly discriminative keys. Technical report, 2005.
- [8] Karl Aberer, Magdalena Puceva, Manfred Hauswirth, and Roman Schmidt. Improving Data Access in P2P Systems. *IEEE Internet Computing*, 6(1):58–67, 2002. ISSN 1089-7801.
- [9] Giuseppe Amato, Paolo Bolettieri, Franca Debole, Fabrizio Falchi, F. Rabitti, and P.Savino. Using MILOS to build an on-line photo album: the PhotoBook. In *SEBD 2006: Proceedings of the Fourtenth Italian Symposium on Advanced Database Systems*, pages 233–240, 2006.
- [10] Giuseppe Amato, Paolo Bolettieri, Franca Debole, Fabrizio Falchi, Fausto Rabitti, and Pasquale Savino. Using MILOS to Build a Multimedia Digital Library Application: The PhotoBook Experience. In *Research and Advanced Technology for Digital Libraries, 10th European Conference, ECDL 2006, Alicante, Spain, September 17-22, 2006, Proceedings*, volume 4172 of *Lecture Notes in Computer Science*, pages 379–390. Springer-Verlag Berlin Heidelberg, 2006.
- [11] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004. ISSN 0360-0300.
- [12] Artur Andrzejak and Zhichen Xu. Scalable, Efficient Range Queries for Grid Information Services. In *P2P '02: Proceedings of the Second International Conference on Peer-to-Peer Computing*, pages 33–40, Washington, DC, USA, September 2002. IEEE Computer Society. ISBN 0-7695-1810-9.

- [13] CURRENT Lab at UC Santa Barbara. Chimera. URL <http://current.cs.ucsb.edu/projects/chimera/>. [Online; accessed 12-November-2006].
- [14] Ricardo A. Baeza-Yates, Walter Cunto, Udi Manber, and Sun Wu. Proximity Matching Using Fixed-Queries Trees. In *CPM '94: Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, pages 198–212, London, UK, 1994. Springer-Verlag. ISBN 3-540-58094-8.
- [15] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in P2P systems. *Communications of the ACM*, 46(2):43–48, February 2003. ISSN 0001-0782.
- [16] Ramrajprabu Balasubramanian, Injong Rhee, and Jaewoo Kang. A scalable architecture for SIP infrastructure using content addressable networks. In *ICC 2005: International Conference on Communications*, volume 2, pages 1314–1318. IEEE Computer Society, 2005.
- [17] David Barkai. *Peer-to-Peer Computing: Technologies for Sharing and Collaborating on the Net*. Intel Press, 2001. ISBN 0970284675.
- [18] Michal Batko. Distributed and Scalable Similarity Searching in Metric Spaces. In *Current Trends in Database Technology - EDBT 2004 Workshops, EDBT 2004 Workshops PhD, DataX, PIM, P2P&DB, and ClustWeb, Heraklion, Crete, Greece, March 14-18, 2004, Revised Selected Papers*, volume 3268 of *Lecture Notes in Computer Science*, pages 44–53, 2004. ISBN 3-540-23305-9.
- [19] Michal Batko, Claudio Gennaro, and Pavel Zezula. A Scalable Nearest Neighbor Search in P2P Systems. In *DBISP2P '04; Databases, Information Systems, and Peer-to-Peer Computing - Second International Workshop, DBISP2P 2004, Toronto, Canada, August 29-30, 2004, Revised Selected Papers*, pages 79–92. Springer-Verlag, 2004.

- [20] Michal Batko, Claudio Gennaro, and Pavel Zezula. SEBD 2004: Proceedings of the Twelfth Italian Symposium on Advanced Database Systems, SEBD 2004, S. Margherita di Pula, Cagliari, Italy, June 21-23, 2004. In *SEBD*, pages 410–417, 2004. ISBN 88-901409-1-7.
- [21] Michal Batko, Claudio Gennaro, and Pavel Zezula. Similarity Grid for Searching in Metric Spaces. In *Peer-to-Peer, Grid, and Service-Oriented in Digital Library Architectures. 6th Thematic Workshop of the EU Network of Excellence DELOS, Cagliari, Italy, June 24-25, 2004, Revised Selected Papers*, volume 3664 of *Lecture Notes in Computer Science*, pages 25–44. Springer-Verlag Berlin Heidelberg, 2004.
- [22] Michal Batko, David Novak, Fabrizio Falchi, and Pavel Zezula. On scalability of the similarity search in the world of peers. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 20, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-428-6.
- [23] Matthias Bender, Sebastian Michel, Peter Triantafillou, Gerhard Weikum, and Christian Zimmer. Improving collection selection with overlap awareness in P2P search engines. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 67–74, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-034-5.
- [24] Matthias Bender, Sebastian Michel, Peter Triantafillou, Gerhard Weikum, and Christian Zimmer. MINERVA: Collaborative P2P Search. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 1263–1266. VLDB Endowment, 2005. ISBN 1-59593-154-6.
- [25] Matthias Bender, Sebastian Michel, Gerhard Weikum, and Christian Zimmer. Challenges of Distributed Search Across Digital Libraries. In Gerhard Weikum, Yannis Ioannidis,

- and Hans-Jrg Schek, editors, *Proceedings of the 8th International Workshop of the DELOS Network of Excellence on Digital Libraries on Future Digital Library Management Systems (System Architecture & Information Access)*, pages 55–59, Dagstuhl, Germany, 2005. Information Society Technologies.
- [26] Matthias Bender, Sebastian Michel, Gerhard Weikum, and Christian Zimmer. The MINERVA Project: Database Selection in the Context of P2P Search. In Gottfried Vossen, Frank Leymann, Peter C. Lockemann, and Wolfrid Stucky, editors, *Datenbanksysteme in Business, Technologie und Web (BTW): 11. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS)*, volume 65 of *Lecture Notes in Informatics*, pages 125–144, Karlsruhe, Germany, March 2005. Gesellschaft fr Informatik. ISBN 3-88579-394-6 ISSN 1617-5468. Acceptance ratio 1:3.
- [27] Matthias Bender, Sebastian Michel, Christian Zimmer, and Gerhard Weikum. Bookmark-driven Query Routing in Peer-to-Peer Web Search. In Jamie Callan, Norbert Fuhr, and Wolfgang Nejdl, editors, *Proceedings of the SIGIR Workshop on Peer-to-Peer Information Retrieval: 27th Annual International ACM SIGIR Conference; SIGIR 2004 P2PIR Workshop*, pages 1–12, Sheffield, UK, 2004. Universitt Duisburg-Essen.
- [28] Matthias Bender, Sebastian Michel, Christian Zimmer, and Gerhard Weikum. Towards Collaborative Search in Digital Libraries Using Peer-to-Peer Technology. In Can Trker, Maristella Agosti, and Hans-Jrg Schek, editors, *Peer-to-peer, grid, and service-orientation in digital library architectures: 6th Thematic Workshop of the EU Network of Excellence DELOS*, volume 3664 of *Lecture Notes in Computer Science*, pages 80–95, Cagliari, Italy, August 2005. Springer. ISBN 3-540-28711-6. Selected, Revised Papers.
- [29] Christian B6hm, Stefan Berchtold, and Daniel A. Keim.

- Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3):322–373, 2001. ISSN 0360-0300.
- [30] Tolga Bozkaya and Meral Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Trans. Database Syst.*, 24(3):361–404, 1999. ISSN 0362-5915.
- [31] Sergey Brin. Near Neighbor Search in Large Metric Spaces. In *VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases*, pages 574–584, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-379-4.
- [32] A. J. Broder. Strategies for efficient incremental nearest neighbor search. *Pattern Recogn.*, 23(1-2):171–178, 1990. ISSN 0031-3203.
- [33] Erik Buchmann and Klemens Böhm. Efficient Evaluation of Nearest-Neighbor Queries in Content-Addressable Networks. In *From Integrated Publication and Information Systems to Virtual Information and Knowledge Environments: Essays Dedicated to Erich J. Neuhold on the Occasion of His 65th Birthday*, volume 3379 of *Lecture Notes in Computer Science*, pages 31–40. Springer-Verlag Berlin Heidelberg, 2005.
- [34] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Commun. ACM*, 16(4):230–236, 1973. ISSN 0001-0782.
- [35] Benjamin Bustos, Gonzalo Navarro, and Edgar Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recogn. Lett.*, 24(14):2357–2366, 2003. ISSN 0167-8655.
- [36] Luis Felipe Cabrera, Michael B. Jones, and Marvin Theimer. Herald: Achieving a Global Event Notification Service. In *HOTOS '01: Proceedings of the Eighth Workshop on Hot*

- Topics in Operating Systems*, page 87, Washington, DC, USA, 2001. IEEE Computer Society.
- [37] M. Castro, P. Druschel, Y. 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.
- [38] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Security for structured peer-to-peer overlay networks. In *OSDI'02: 5th Symposium on Operating Systems Design and Implementaion*, December 2002.
- [39] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Exploiting Network Proximity in Distributed Hash Tables. In Ozalp Babaoglu, Ken Birman, and Keith Marzullo, editors, *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, pages 52–55, June 2002.
- [40] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony I. T. Rowstron. Topology-Aware Routing in Structured Peer-to-Peer Overlay Networks. In *Future Directions in Distributed Computing*, volume 2584/2001 of *Lecture Notes in Computer Science*, pages 103–107. Springer-Verlag, 2003. ISBN 3-540-00912-4.
- [41] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. One ring to rule them all: service discovery and binding in structured peer-to-peer overlay networks. In *EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop: beyond the PC*, pages 140–145, New York, NY, USA, 2002. ACM Press.
- [42] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony I.T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, October 2002. ISSN 0733-8716.

- [43] Miguel Castro, Michael B. Jones, Anne-Marie Kermarrec, Antony Rowstron, Marvin Theimer, Helen Wang, and Alec Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *INFOCOM 2003: Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1510–1520. IEEE, 2003.
- [44] Edgar Chávez, J. L. Marroquín, and Ricardo Baeza-Yates. Spaghettis: An Array Based Algorithm for Similarity Queries in Metric Spaces. In *SPIRE '99: Proceedings of the String Processing and Information Retrieval Symposium & International Workshop on Groupware*, page 38, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0268-7.
- [45] Edgar Chávez, José L. Marroquín, and Gonzalo Navarro. Fixed Queries Array: A Fast and Economical Data Structure for Proximity Searching. *Multimedia Tools Appl.*, 14(2): 113–135, 2001. ISSN 1380-7501.
- [46] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Computing Surveys (CSUR)*, 33(3):273–321, 2001. ISSN 0360-0300.
- [47] Sergey Chernov, Pavel Serdyukov, Matthias Bender, Sebastian Michel, Gerhard Weikum, and Christian Zimmer. Database Selection and Result Merging in P2P Web Search. In *Third International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2005)*, pages 1–14, Trondheim, Norway, 2005. Springer. Acceptance rate 1:3.
- [48] Paolo Ciaccia, Danilo Montesi, Wilma Penzo, and Alberto Trombetta. Imprecision and User Preferences in Multimedia Queries: A Generic Algebraic Approach. In Klaus-Dieter Schewe and Bernhard Thalheim, editors, *FoIKS*, volume 1762

- of *Lecture Notes in Computer Science*, pages 50–71. Springer, 2000. ISBN 3-540-67100-5.
- [49] Paolo Ciaccia and Marco Patella. The M2-tree: Processing Complex Multi-Feature Queries with Just One Index. In *DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries*, 2000.
- [50] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 426–435, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1-55860-470-7.
- [51] Paolo Ciaccia, Marco Patella, and Pavel Zezula. Processing Complex Similarity Queries with Distance-Based Access Methods. In Hans-Jörg Schek, Fèlix Saltor, Isidro Ramos, and Gustavo Alonso, editors, *EDBT*, volume 1377 of *Lecture Notes in Computer Science*, pages 9–23. Springer, 1998. ISBN 3-540-64264-1.
- [52] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 2000, Proceedings*, volume 2009 of *Lecture Notes in Computer Science*, pages 46–66. Springer-Verlag, 2001.
- [53] Kenneth L. Clarkson. Nearest neighbor queries in metric spaces. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 609–617, New York, NY, USA, 1997. ACM Press. ISBN 0-89791-888-6.

- [54] Jon Crowcroft and Ian Pratt. *Peer to peer: peering into the future*. Springer-Verlag New York, Inc., New York, NY, USA, 2002. ISBN 3-540-00165-4.
- [55] Anwitaman Datta, Manfred Hauswirth, Renault John, Roman Schmidt, and Karl Aberer. Range Queries in Trie-Structured Overlays. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 57–66, 2005.
- [56] Robert Devine. Design and Implementation of DDH: A Distributed Dynamic Hashing Algorithm. In *FODO '93: Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, pages 101–114, London, UK, 1993. Springer-Verlag. ISBN 3-540-57301-1.
- [57] Vlastislav Dohnal, Claudio Gennaro, Pasquale Savino, and Pavel Zezula. D-Index: Distance Searching Index for Metric Data Sets. *Multimedia Tools Appl.*, 21(1):9–33, 2003. ISSN 1380-7501.
- [58] Peter Druschel and Antony Rowstron. PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility. In *HOTOS '01: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, page 75, Washington, DC, USA, 2001. IEEE Computer Society.
- [59] Jrg Eberspcher and Rdiger Schollmeier. First and Second Generation of Peer-to-Peer-Systems. In *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*, chapter 5, pages 35–56. Springer-Verlag Berlin Heidelberg, 2005.
- [60] Ronald Fagin. Combining Fuzzy Information from Multiple Systems. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, 1996, Montreal, Canada*, pages 216–226. ACM Press, 1996. ISBN 0-89791-781-2.

- [61] Ronald Fagin. Fuzzy Queries in Multimedia Database Systems. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, pages 1–10. ACM Press, 1998. ISBN 0-89791-996-3.
- [62] Ronald Fagin. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences*, 58(1): 83–99, 1999. ISSN 0022-0000.
- [63] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal Aggregation Algorithms for Middleware. In *PODS*. ACM, 2001. ISBN 1-58113-361-8.
- [64] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal Aggregation Algorithms for Middleware. *CoRR*, cs.DB/0204046, 2002.
- [65] Fabrizio Falchi, Claudio Gennaro, and Pavel Zezula. A Content-Addressable Network for Similarity Search in Metric Spaces. In Gianluca Moro, Sonia Bergamaschi, and Aris M. Ouksel, editors, *DBISP2P '05: Proceedings of the the 2nd International Workshop on Databases, Information Systems and Peer-to-Peer Computing, Trondheim, Norway*, volume 4125 of *Lecture Notes in Computer Science*, pages 98–110. Springer, 2005.
- [66] Fabrizio Falchi, Claudio Gennaro, and Pavel Zezula. Nearest Neighbor Search in Metric Spaces through Content-Addressable Networks. *Information Processing & Management*, 43(3):665–683, May 2007.
- [67] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *J. Intell. Inf. Syst.*, 3(3-4):231–262, 1994. ISSN 0925-9902.

- [68] Hassan M. Fattah. *P2p: How Peer-to-Peer Technology Is Revolutionizing the Way We Do Business*. Dearborn Financial Publishing, Inc., 2002. ISBN 0793148782.
- [69] Ian Foster. Internet Computing and the Emerging Grid. *Nature*, 408(6815), 2000.
- [70] Ian Foster. What is the Grid? - a three point checklist. *GRIDtoday*, 1(6), July 2002.
- [71] Ian T. Foster and Adriana Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In *IPTPS 2003: Peer-to-Peer Systems II, Second International Workshop, Berkeley, CA, USA, February 21-22, 2003, Revised Papers*, volume 2735 of *Lecture Notes in Computer Science*, pages 118–128, 2003.
- [72] Prasanna Ganesan, Beverly Yang, and Hector Garcia-Molina. One torus to rule them all: multi-dimensional queries in P2P systems. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 19–24, New York, NY, USA, 2004. ACM Press.
- [73] Claudio Gennaro, Pasquale Savino, and Pavel Zezula. Similarity search in metric databases through hashing. In *MULTIMEDIA '01: Proceedings of the 2001 ACM workshops on Multimedia*, pages 1–5, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-395-2.
- [74] Brighten Godfrey, Karthik Lakshminarayanan, Sonesh Surana, Richard M. Karp, and Ion Stoica. Load Balancing in Dynamic Structured P2P Systems. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 2007–2016, 2004. IEEE.
- [75] Robert L. Goldstone and Son Ji Yun. Similarity. In K. Holyoak and R. Morrison, editors, *Cambridge Handbook*

- of Thinking and Reasoning*, pages 13–36. Cambridge University Press, Cambridge, 2005. URL <http://cognitron.psych.indiana.edu/rgoldsto/papers.html>.
- [76] Li Gong. JXTA: A Network Programming Environment. *IEEE Internet Computing*, 5(3):88–95, 2001. ISSN 1089-7801.
- [77] Sudipto Guha, H. V. Jagadish, Nick Koudas, Divesh Srivastava, and Ting Yu. Approximate XML joins. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 287–298, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-497-5.
- [78] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Optimizing Multi-Feature Queries for Image Databases. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 419–428, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-715-3.
- [79] Andreas Haeberlen, Jeff Hoyer, Alan Mislove, and Peter Druschel. Consistent Key Mapping in Structured Overlays. Technical Report TR05-456, Rice University, Department of Computer Science, August 2005.
- [80] James Hafner, Harpreet S. Sawhney, Will Equitz, Myron Flickner, and Wayne Niblack. Efficient Color Histogram Indexing for Quadratic Form Distance Functions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(7):729–736, 1995. ISSN 0162-8828.
- [81] Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon Thau Loo, Scott Shenker, and Ion Stoica. Complex Queries in DHT-based Peer-to-Peer Networks. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *Peer-to-Peer Systems. First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, volume 2429 of *Lecture Notes in Computer Sci-*

- ence, pages 242–259. Springer-Verlag Berlin Heidelberg, 2002. ISBN 3-540-44179-4.
- [82] Sugata Hazarika and Don Towsley. Delay analysis of application level multicast on content addressable networks. In *GLOBECOM '04: Proceeding of the Global Telecommunications Conference, 2004*, volume 2, pages 1271–1277 Vol.2. IEEE, 2004.
- [83] Gísli R. Hjaltason and Hanan Samet. Distance Browsing in Spatial Databases. *ACM Transactions on Database Systems (TODS)*, 24(2):265–318, 1999.
- [84] Gisli R. Hjaltason and Hanan Samet. Index-driven similarity search in metric spaces (Survey Article). *ACM Trans. Database Syst.*, 28(4):517–580, 2003. ISSN 0362-5915.
- [85] Josef F. Huber. *Peer-to-peer networking in mobile communications based on SIP*. Plenum Press, New York, NY, USA, 2004. ISBN 0-306-48190-1.
- [86] D. P. Huttenlocher, G. A. Klanderman, and W. A. Rucklidge. Comparing Images Using the Hausdorff Distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9):850–863, 1993. ISSN 0162-8828.
- [87] ISO/IEC. Information technology - Multimedia content description interfaces., 2002. 15938.
- [88] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30(2):364–397, 2005. ISSN 0362-5915.
- [89] William James. *The Principles of Psychology*. Dover, 1890/1950.
- [90] Michael B. Jones, Marvin Theimer, Helen J. Wang, and Alec Wolman. Unexpected Complexity: Experiences Tuning and

- Extending CAN. Technical Report MSR-TR-2002-118, Microsoft Research, One Microsoft Way, Redmond, WA 98052, 2002.
- [91] M. Frans Kaashoek and David R. Karger. Koorde: A Simple Degree-Optimal Distributed Hash Table. In *IPTPS 2003: Peer-to-Peer Systems II, Second International Workshop, Berkeley, CA, USA, February 21-22, 2003, Revised Papers*, pages 98–107, 2003.
- [92] Iraj Kalantari and Gerard McDonald. A Data Structure and an Algorithm for the Nearest Point Problem. *IEEE Trans. Software Eng.*, 9(5):631–634, 1983.
- [93] Jonas S. Karlsson, Witold Litwin, and Tore Risch. LH*LH: A scalable High Performance Data Structure for Switched Multicomputers. In Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin, editors, *EDBT*, volume 1057 of *Lecture Notes in Computer Science*, pages 573–591. Springer, 1996. ISBN 3-540-61057-X.
- [94] Farnoush Banaei Kashani and Cyrus Shahabi. SWAM: a family of access methods for similarity-search in peer-to-peer data networks. In David Grossman, Luis Gravano, ChengXiang Zhai, Otthein Herzog, and David A. Evans, editors, *CIKM*, pages 304–313. ACM, 2004. ISBN 1-58113-874-1.
- [95] Kostas Katrinis and Martin May. Application-Layer Multicast. In *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*, chapter 11, pages 157–170. Springer-Verlag Berlin Heidelberg, 2005.
- [96] John L. Kelley. *General Topology*. Van Nostrand Reinhold, New York Heidelberg Berlin, 1955.
- [97] Brigitte Kröll and Peter Widmayer. Distributing a search tree among a growing number of processors. In *SIGMOD '94*:

- Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 265–276, New York, NY, USA, 1994. ACM Press. ISBN 0-89791-639-5.
- [98] John Kubiawicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gum-madi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and Ben Zhao. OceanStore: an architecture for global-scale per-sistent storage. In *ASPLOS-IX: Proceedings of the ninth in-ternational conference on Architectural support for program-ming languages and operating systems*, pages 190–201, New York, NY, USA, 2000. ACM Press. ISBN 1-58113-317-0.
- [99] Per-Ake Larson. Dynamic hash tables. *Commun. ACM*, 31(4):446–457, 1988. ISSN 0001-0782.
- [100] Bo Leuf. *Peer to Peer: Collaboration and Sharing over the Internet*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. ISBN 0201767325.
- [101] Vladimir I. Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Infor-mation Transmission*, 1:8–17, 1965.
- [102] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.
- [103] Jinyang Li, Jeremy Stribling, Thomer M. Gil, Robert Morris, and M. Frans Kaashoek. Comparing the Performance of Dis-tributed Hash Tables Under Churn. In *Peer-to-Peer Systems III, Third International Workshop, IPTPS 2004, La Jolla, CA, USA, February 26-27, 2004, Revised Selected Papers*, volume 3279 of *Lecture Notes in Computer Science*, pages 87–99. Springer-Verlag Berlin Heidelberg, 2004.
- [104] Qiao Lian, Zheng Zhang, Shaomei Wu, and Ben Y. Zhao. Z-Ring: Fast Prefix Routing via a Low Maintenance Member-ship Protocol. In *ICNP '05: Proceedings of the 13TH IEEE*

- International Conference on Network Protocols (ICNP'05)*, pages 132–146, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2437-0.
- [105] W. Litwin, M.-A. Neimat, G. Lev, S. Ndiaye, and T. Seck. LH*s: a high-availability and high-security scalable distributed data structure. In *RIDE '97: Proceedings of the 7th International Workshop on Research Issues in Data Engineering (RIDE '97) High Performance Database Management for Large-Scale Applications*, page 141, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-7849-6.
- [106] Witold Litwin. Linear Hashing: A New Tool for File and Table Addressing. In *VLDB'80: Sixth International Conference on Very Large Data Bases, October 1-3, 1980, Montreal, Quebec, Canada, Proceedings*, pages 212–223. IEEE Computer Society, 1980.
- [107] Witold Litwin. Linear Hashing: a new tool for file and table addressing. pages 570–581, 1988.
- [108] Witold Litwin. Scalable Distributed Data Structures, 2006. URL <http://ceria.dauphine.fr/SDDS-bibliographie.html>. [Online; accessed 4-December-2006].
- [109] Witold Litwin and Marie-Anne Neimat. High-Availability LH* Schemes with Mirroring. In *CoopIS'96: First IFCIS International Conference on Cooperative Information Systems*, pages 196–205, Los Alamitos, CA, USA, 1996. IEEE Computer Society.
- [110] Witold Litwin and Marie-Anne Neimat. k-RP*S: A Scalable Distributed Data Structure for High-Performance Multi-Attribute Access. In *PDIS 1996: Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems*, pages 120–131, 1996.
- [111] Witold Litwin, Marie-Anne Neimat, and Donovan A. Schneider. LH* - Linear Hashing for Distributed Files. In Peter

- Buneman and Sushil Jajodia, editors, *SIGMOD Conference*, pages 327–336. ACM Press, 1993.
- [112] Witold Litwin, Marie-Anne Neimat, and Donovan A. Schneider. RP*: A Family of Order Preserving Scalable Distributed Data Structures. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 342–353. Morgan Kaufmann, 1994. ISBN 1-55860-153-8.
- [113] Witold Litwin, Marie-Anne Neimat, and Donovan A. Schneider. LH* - A Scalable, Distributed Data Structure. *ACM Transactions on Database Systems (TODS)*, 21(4):480–525, 1996.
- [114] Ratul Mahajan, Miguel Castro, and Antony Rowstron. Controlling the Cost of Reliability in Peer-to-peer Overlays. In *IPTPS'03: Proceedings of 2nd International Workshop on Peer-to-Peer Systems*, February 2003.
- [115] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-485-1.
- [116] G. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed Hashing in a Small World. In *USITS 2003: Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, 2003.
- [117] Amelie Marian. Detecting Changes in XML Documents. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, page 41, Washington, DC, USA, 2002. IEEE Computer Society.
- [118] Petar Maymounkov and David Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In

- Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65. Springer-Verlag Berlin Heidelberg, 2002.
- [119] Sebastian Michel, Matthias Bender, Peter Triantafillou, and Gerhard Weikum. IQN Routing: Integrating Quality and Novelty in P2P Querying and Ranking. In Yannis Yoannidis, Marc H. Scholl, Joachim W. Schmidt, Florian Matthes, Michael Hatzopoulos, Klemens Bhm, Alfons Kemper, Torsten Grust, and Christian Bhm, editors, *Advances in Database Technology - EDBT 2006, Proceedings of the 10th International Conference on Extending Database Technology (EDBT06)*, volume 3896 of *LNCS*, pages 149–166, Munich, Germany, March 2006. Springer. ISBN 3-540-32960-9.
- [120] Sebastian Michel, Matthias Bender, Peter Triantafillou, Gerhard Weikum, and Christian Zimmer. P2P Web Search with MINERVA: How do you want to search tomorrow? (Demo). Technical Report DELIS-TR-0293, University of Paderborn, Heinz Nixdorf Institute, Grenoble, France, 2005.
- [121] Sebastian Michel, Peter Triantafillou, and Gerhard Weikum. KLEE: A Framework for Distributed Top-k Query Algorithms. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 637–648. ACM, 2005.
- [122] María Luisa Micó, José Oncina, and Enrique Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESAs) with linear preprocessing time and memory requirements. *Pattern Recognitions Letters*, 15 (1):9–17, 1994. ISSN 0167-8655.
- [123] Michael Miller. *Discovering P2P*. SYBEX Inc., Alameda, CA, USA, 2001. ISBN 0782140181.
- [124] Dejan S. Milojevic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and

- Zhichen Xu. Peer-to-Peer Computing. Technical Report HPL-2002-57 (R.1), HP Laboratories Palo Alto, 2003.
- [125] Alan Mislove, Andreas Haeberlen, Ansley Postm, and Peter Druschel. ePOST. In *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*, chapter 12, pages 171–192. Springer-Verlag Berlin Heidelberg, 2005.
- [126] H. Garcia Molina and B. Kogan. Node autonomy in distributed systems. In *DPDS '88: Proceedings of the first international symposium on Databases in parallel and distributed systems*, pages 158–166, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press. ISBN 0-8186-0893-5.
- [127] Anirban Mondal, Yi Lifu, and Masaru Kitsuregawa. P2PR-Tree: An R-Tree-Based Spatial Index for Peer-to-Peer Environments. In *Current Trends in Database Technology. EDBT 2004 Workshops: EDBT 2004 Workshops PhD, DataX, PIM, P2P&DB, and ClustWeb, Heraklion, Crete, Greece, March 14-18, 2004. Revised Selected Papers*, volume 3268 of *Lecture Notes in Computer Science*, pages 516–525. Springer-Verlag Berlin Heidelberg, 2004.
- [128] Dana T. Moore and John Hebel. *Peer-to-Peer: Tap into the Power of the Internet*. McGraw-Hill Professional, 2001. ISBN 0072192844.
- [129] Enrico Nardelli, Fabio Barillari, and Massimo Pepe. Distributed searching of multi-dimensional data: a performance evaluation study. *J. Parallel Distrib. Comput.*, 49(1):111–134, 1998. ISSN 0743-7315.
- [130] Gonzalo Navarro. Searching in Metric Spaces by Spatial Approximation. In *SPIRE '99: Proceedings of the String Processing and Information Retrieval Symposium & International Workshop on Groupware*, page 141, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0268-7.

- [131] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001. ISSN 0360-0300.
- [132] Gonzalo Navarro. Searching in metric spaces by spatial approximation. *The VLDB Journal*, 11(1):28–46, 2002. ISSN 1066-8888.
- [133] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.
- [134] Surya Nepal and M. V. Ramakrishna. Query Processing Issues in Image(Multimedia) Databases. In *ICDE '99: Proceedings of the 15th International Conference on Data Engineering*, page 22, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0071-4.
- [135] Heiko Niedermayer, Simon Rieche, Klaus Wehrle, and Georg Carle. On the Distribution of Nodes in Distributed Hash Tables. In *Proceedings of Workshop Peer-to-Peer-Systems and -Applications, KiVS 2005*, Kaiserslautern, Germany, 2005.
- [136] David Novak and Pavel Zezula. M-Chord: a scalable distributed similarity search structure. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 19, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-428-6.
- [137] National Institute of Standards and Technology. *FIPS PUB 180-1: Secure Hash Standard*. National Institute for Standards and Technology, Gaithersburg, MD, USA, April 1995. Supersedes FIPS PUB 180 1993 May 11.
- [138] Andy Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001. ISBN 059600110X.

- [139] Tim O'Reilly. Remaking the Peer-to-Peer Meme. In Andy Oram, editor, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, pages 29–40. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.
- [140] Maria Papadopouli and Henning Schulzrinne. *Peer-to-Peer Computing for Mobile Networks: Information Discovery and Dissemination*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. ISBN 0387244271.
- [141] Odysseas Papapetrou, Sebastian Michel, Matthias Bender, and Gerhard Weikum. On the Usage of Global Document Occurrences in Peer-to-Peer Information Systems. In Robert Meersman, Zahir Tari, Mohand-Said Hacid, John Mylopoulos, Barbara Pernici, zalp Babaoglu, Hans-Arno Jacobsen, Joseph P. Loyall, Michael Kifer, and Stefano Spaccapietra, editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005*, volume 3760 of *Lecture Notes in Computer Science*, pages 310–328, Agia Napa, Cyprus, 2005. Springer. ISBN 3-540-29736-7. Acceptance Rate:.
- [142] Michal Parnas and Dana Ron. Testing metric properties. *Information and Computation*, 187(2):155–195, 2003.
- [143] R. Rammal, G. Toulouse, and M. A. Virasoro. Ultrametricity for physicists. *Reviews of Modern Physics*, 58(3):765–788, Jul 1986.
- [144] Sylvia Ratnasamy. *A Scalable Content-Addressable Network*. PhD thesis, University of California, Berkeley, 2002.
- [145] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-411-8.

- [146] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott Shenker. Application-Level Multicast Using Content-Addressable Networks. In Jon Crowcroft and Markus Hofmann, editors, *Networked Group Communication. Third International COST264 Workshop, NGC 2001, London, UK, November 7-9, 2001*, volume 2233 of *Lecture Notes in Computer Science*, pages 14–29, London, UK, 2001. Springer-Verlag. ISBN 3-540-42824-0.
- [147] Thomas Reidemeister, Paul A.S. Ward, Klemens Bohm, and Erik Buchmann. Malicious behaviour in content-addressable peer-to-peer networks. In *Communication Networks and Services Research Conference, 2005. Proceedings of the 3rd Annual*, pages 319–326. IEEE, 2005.
- [148] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiawicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. OpenDHT: a public DHT service and its uses. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 73–84, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-009-4.
- [149] Simon Rieche, Heiko Niedermayer, Stefan Gtz, and Klaus Wehrle. Reliability and Load Balancing in Distributed Hash Tables. In *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*, chapter 9, pages 119–135. Springer-Verlag Berlin Heidelberg, 2005.
- [150] Simon Rieche, Leo Petrak, and Klaus Wehrle. A Thermal-Dissipation-Based Approach for Balancing Data Load in Distributed Hash Tables. In *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN'04)*, pages 15–23, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2260-2.
- [151] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer

- storage utility. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 188–201, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-389-8.
- [152] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag. ISBN 3-540-42800-3.
- [153] O.D. Sahin, D. Agrawal, and A.E. Abbadi. Techniques for Efficient Routing and Load Balancing in Content-Addressable Networks. In *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, pages 67–74, 2005.
- [154] Phillipe Salembier and Thomas Sikora. *Introduction to MPEG-7: Multimedia Content Description Interface*. John Wiley & Sons, Inc., New York, NY, USA, 2002. ISBN 0471486787.
- [155] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4): 277–288, 1984. ISSN 0734-2071.
- [156] Hanan Samet. *The design and analysis of spatial data structures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990. ISBN 0-201-50255-0.
- [157] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Computer Graphics and Geometric Modeling. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006. ISBN 0123694469.
- [158] Thomas Seidl and Hans-Peter Kriegel. Efficient User-Adaptable Similarity Search in Large Multimedia Databases.

- In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 506–515, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1-55860-470-7.
- [159] C. Severance, S. Pramanik, and P. Wolberg. Distributed linear hashing and parallel projection in main memory databases. In *Proceedings of the sixteenth international conference on Very large databases*, pages 674–682, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc. ISBN 0-55860-149-X.
- [160] Dennis Shasha and Tsong-Li Wang. New techniques for best-match retrieval. *ACM Trans. Inf. Syst.*, 8(2):140–158, 1990. ISSN 1046-8188.
- [161] Clay Shirky. Listening to Napster. In Andy Oram, editor, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001. ISBN 059600110X.
- [162] Tyron Stading, Petros Maniatis, and Mary Baker. Peer-to-Peer Caching Schemes to Address Flash Crowds. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 203–213, London, UK, 2002. Springer-Verlag. ISBN 3-540-44179-4.
- [163] Ralf Steinmetz and Klaus Wehrle. Peer-to-Peer-Networking & -Computing. Aktuelles Schlagwort. *Informatik Spektrum*, 27(1):51–54, 2004.
- [164] Ralf Steinmetz and Klaus Wehrle, editors. *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Heidelberg, Germany, nov 2005. ISBN 354029192X.
- [165] Ralf Steinmetz and Klaus Wehrle. What Is This Peer-to-Peer About? In *Peer-to-Peer Systems and Applications*, volume

- 3485 of *Lecture Notes in Computer Science*, chapter 2, pages 9–16. Springer-Verlag Berlin Heidelberg, 2005.
- [166] Ion Stoica, Daniel Adkins, Sylvia Ratnasamy, Scott Shenker, Sonesh Surana, and Shelley Zhuang. Internet Indirection Infrastructure. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 191–202, London, UK, 2002. Springer-Verlag. ISBN 3-540-44179-4.
- [167] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-411-8.
- [168] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1):17–32, 2003.
- [169] Ramesh Subramanian and Brian D. Goodman. *Peer to Peer Computing: The Evolution of a Disruptive Technology*. Idea Group Publishing, Hershey, PA, USA, 2005. ISBN 1591404290.
- [170] Torsten Suel, Chandan Mathur, Jo wen Wu, Jiangong Zhang, Alex Delis, Mehdi Kharrazi, Xiaohui Long, and Kulesh Shanmugasundaram. ODISSEA: A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval. In *WWW (Posters)*, 2003.
- [171] Sonesh Surana, Brighten Godfrey, Karthik Lakshminarayanan, Richard Karp, and Ion Stoica. Load balancing in dynamic structured peer-to-peer systems. In *Performance*

- Evaluation*, volume 63, pages 217–240. P2P Computing Systems, March 2006.
- [172] D. Takemoto, S. Tagashira, and S. Fujita. Distributed algorithms for balanced zone partitioning in content-addressable networks. In *Parallel and Distributed Systems, 2004. ICPADS 2004. Proceedings. Tenth International Conference on*, pages 377–384, 2004.
- [173] Chunqiang Tang and Sandhya Dwarkadas. Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval. In *NSDI 2004: 1st Symposium on Networked Systems Design and Implementation, March 29-31, 2004, San Francisco, California, USA, Proceedings*, pages 211–224, 2004.
- [174] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 175–186, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-735-4.
- [175] Pastry Project Team. FreePastry, 2006. URL <http://freepastry.rice.edu/FreePastry>. [Online; accessed 12-November-2006].
- [176] Jeffrey K. Uhlmann. Satisfying General Proximity/Similarity Queries with Metric Trees. *Information Processing Letters*, 40(4):175–179, 1991.
- [177] Enrique Vidal. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognition Letters*, 4(3):145–157, 1986. ISSN 0167-8655.
- [178] Enrique Vidal. New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (AESA). *Pattern Recogn. Lett.*, 15(1):1–7, 1994. ISSN 0167-8655.

- [179] Radek Vingralek, Yuri Breitbart, and Gerhard Weikum. Distributed file organization with scalable cost/performance. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 253–264, New York, NY, USA, 1994. ACM Press. ISBN 0-89791-639-5.
- [180] Radek Vingralek, Yuri Breitbart, and Gerhard Weikum. Snowball: Scalable Storage on Networks of Workstations with Balanced Load. *Distrib. Parallel Databases*, 6(2):117–156, 1998. ISSN 0926-8782.
- [181] Klaus Wehrle, Stefan Gtz, and Simon Rieche. Distributed Hash Tables. In *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*, chapter 7, pages 79–93. Springer-Verlag Berlin Heidelberg, 2005.
- [182] Klaus Wehrle, Stefan Gtz, and Simon Rieche. Selected DHT Algorithms. In *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*, chapter 8, pages 95–117. Springer-Verlag Berlin Heidelberg, 2005.
- [183] Klaus Wehrle and Ralf Steinmetz. Introduction. In *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*, chapter 1, pages 1–5. Springer-Verlag Berlin Heidelberg, 2005.
- [184] Wikipedia. Kademia — Wikipedia, The Free Encyclopedia, 2007. URL <http://en.wikipedia.org/w/index.php?title=Kademia>. [Online; accessed 9-January-2007].
- [185] Wikipedia. Napster Network — Wikipedia, The Free Encyclopedia, 2007. URL <http://en.wikipedia.org/w/index.php?title=Napster&oldid=97795564>. [Online; accessed 8-January-2007].
- [186] Wikipedia. Peer-to-peer — Wikipedia, The Free Encyclopedia, 2007. URL <http://en.wikipedia.org/w/index.php?title=Peer-to-peer&oldid=99127751>. [Online; accessed 8-January-2007].

- [187] Wikipedia. Wikipedia — Wikipedia, The Free Encyclopedia, 2007. URL <http://en.wikipedia.org/w/index.php?title=Wikipedia&oldid=99305653>. [Online; accessed 8-January-2007].
- [188] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA '93: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 311–321, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics. ISBN 0-89871-313-7.
- [189] Peter N. Yianilos. Excluded middle vantage point forests for nearest neighbor search. Technical report, NEC Research Institute, Princeton, NJ, July 1999. Presented at the Sixth DIMACS Implementation Challenge: Near Neighbor Searches workshop, January 15, 1999.
- [190] Cui Yu, Beng Chin Ooi, Kian-Lee Tan, and H. V. Jagadish. Indexing the Distance: An Efficient Method to KNN Processing. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 421–430. Morgan Kaufmann Publishers Inc., 2001.
- [191] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search. The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer Science + Business Media, Inc., 233 Spring Street, New York, NY 10013, USA, 2006. ISBN 0387291466.
- [192] Jiangong Zhang and T. Suel. Efficient Query Evaluation on Large Textual Collections in a Peer-to-Peer Environment. In *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, pages 225–233, 2005.
- [193] Ben Y. Zhao, John D. Kubiawicz, and Anthony D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB//CSD-01-1141,

University of California Berkeley, Electrical Engineering and Computer Science Department, Berkeley, CA, USA, April 2001.

- [194] Ben Y. Zhao, John D. Kubiawicz, and Anthony D. Joseph. Tapestry: a fault-tolerant wide-area application infrastructure. *SIGCOMM Comput. Commun. Rev.*, 32(1):81–81, 2002. ISSN 0146-4833.
- [195] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiawicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 11–20, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-370-7.
- [196] M. Tamer zsu and Patrick Valduriez. Distributed and parallel database systems. *ACM Computing Surveys (CSUR)*, 28(1): 125–128, 1996. ISSN 0360-0300.

Index

- \mathcal{A}_0 , 6
- μ Torrent, 44
- k NN, *see* Nearest Neighbor query
- 1-hop route check, 69
- 1-hop volume check, 66
- 1-lookahead approach, 47
- ad hoc communities, 19
- Address Search Tree, 53
- adjacent, *see* zone, adjacent
- AESA, 13
- aMule, 43
- application-level multicast, 42, 49, 74
- AST, *see* Address Search Tree 53
- Azureus, 44
- ball partitioning, 12, 52
- Bayeux, 42, 74
- Bisector Tree, 12
- BitSpirit, 44
- BitTorrent, 44
- BKT, *see* Burkhard-Keller Tree
- Bloom Circle Threshold Algorithm, 50
- Bloom Petal Threshold Algorithm, 50
- bootstrap, 66
- BST, *see* Bisector Tree
- Burkhard-Keller Tree, 12
- caching, 70
- CAN, 50, 52, 63–75
- CASIP, 75
- Chimera, 42
- Chord, 36–39, 44, 50, 52
- complex queries, 49
- complex similarity queries, 5
- Content-Addressable Network, *see* CAN
- contraction, *see* mapping, contraction, 78
- D-Index, 13
- decentralized
 - resource usage, 21
 - self-organization, 22
- DHT, *see* Distributed Hash Table, 25, 30, 143
- dimensionality curse, 2
- direct storage, 33
- distance, 7
 - L_1 , 9
 - asymmetric, 8
 - browsing, 6
 - chessboard, *see* distance, L_∞
 - City-Block, *see* distance, L_1
 - Euclidean, 9, 10
 - Hausdorff, 11
 - Levenshtein, *see* distance, edit
 - Manhattan, *see* distance, L_1
 - maximum, *see* distance, L_∞
 - metric, *see* metric distance

- quadratic form, 10
- distance, edit, 10
- distance, tree edit, 11
- distributed
 - system, 20
 - systems, 21
- Distributed Hash Table, 15, 50, 63
- distributed indexing, 25
- Distributed Linear Hashing, 18
- Distributed Random Tree, 18
- Distributed Threshold Algorithm, 50
- DRT, *see* Distributed Random Tree
- dynamic hashing, 18

- edit distance, *see* distance, edit, 98
- eDonkey2000, 43
- email, 41
- eMule, 43
- ePost, 41
- exact match, 4, 51
- Excluded middle Vantage Point Forest, 12

- Fixed Queries Array, 12
- Fixed Queries Tree, 12
- FQA, *see* Fixed Queries Array
- FQT, *see* Fixed Queries Tree
- Freenet, 24
- FreePastry, 41
- function
 - pseudo-metric, 8
 - quasi-metric, 8
 - super-metric, 8
 - ultra-metric, 8
- generalized hyperplane partitioning, 12, 52
- Generalized Hyperplane Tree, 12, 54

- Geometric Near-neighbor Access Tree, 13
- GHT, *see* Generalized Hyperplane Tree
- GHT*, 53–57
- GNAT, *see* Geometric Near-neighbor Access Tree
- Gnutella, 20, 24, 28
- Google
 - Image Labeler, xv
 - Image Search, xv
- Grid computing, 19, 59, 60
- hot-spot, 17, 70

- Image Adjustment Messages, 17
- Incremental Nearest Neighbor, 6
- incremental selection technique, 79
- incremental similarity search, 6
- information retrieval, 50
- INN, *see* Incremental Nearest Neighbor
- inverted lists, 50

- JXTA, 24

- k-RP*S, 18
- Kad, 43
- KadC, 43
- Kademlia, 43–44, 46
- Kazaa, 20
- Khashmir, 44
- KLEE, 52
- KTorrent, 44

- LAESA, *see* Linear AESA, 81
- latent semantic indexing, 50
- leaf set, 40
- Levenshtein distance, *see* distance, edit

- LH*, *see* Distributed Linear Hashing
- LH*s, 18
- Linear AESA, 13
- linear hashing, 18
- load balancing, 31, 49, 70
- lookup problem, 25

- M-CAN, 70–74
- M-Chord, 52, 57–59
- M-tree, 6, 13
- Mainline, 44
- mapping
 - contraction, 78
 - function, 78
 - nonexpansive, 78
 - strict contraction, 78
- mapping function, 78
- MCAN, 77–94
- membership protocol, 43
- metric, 7
 - distance, 7
 - distance measures, 9
 - function, 7
 - space, 7–9
- Metric Content-Addressable Network,
 - see* MCAN
- MINERVA, 49
- Minkowski distance, 9
- MLDonkey, 43
- Multi Vantage Point Tree, 13
- multicast, 49
- MVPT, *see* Multi Vantage Point Tree

- Napster, 19, 21, 24, 27
- Nearest Neighbor query, 4, 6, 51, 81, 88
- NeP4B, xviii

- Netscape, 18
- network proximity metric, 40
- nonexpansive, *see* mapping, nonexpansive

- OceanStore, 42
- OpenDHT Project, 33
- overlay network, *see* distributed system, 25
- overlay system, *see* distributed system, 25
- Overnet, 43

- P-Grid, 19, 51
- P2P, *see* Peer-to-Peer
- P2PR-tree, 52
- partition tree, 65
- Pastry, 39–42, 74
- peer, 24
- Peer-to-Peer, 19–61
 - first generation, 26
 - flooding, 25
 - hybrid, 25
 - pure, 24
 - second generation, 26
 - server-based, 22, 25
 - structured, 25, 26, 29
 - system, 20
 - unstructured, 26–29, 50
- pivot, 12, 54, 55, 58, 77, 78
 - filtering, *see* pivoted filtering selection, 79
- pivoted filtering, 80
- point query, *see* exact match
- prefix routing scheme, 51
- proactive recovery mechanisms, 34
- pSearch, 50

- quadratic form distance, *see* distance, quadratic form

- random graphs, 51
- range partitioning, 39
- range query, 4, 39, 51
- replication, 70
- RevConnect, 43
- routing, 32
 - algorithm, 32
 - metric, 32
 - table, 43
- RP*, 18
- RQ, *see* range query

- SAPIR, xvii
- SAT, *see* Spatial Approx. Tree
- Scalable and Distributed Data Structure, 15–19
- SCRAP, 39
- SCRIBE, 74
- Scribe, 41
- SDDS, *see* Scalable and Distributed Data Structure
- search
 - problem, 3
- self-organization, 22, 50
- self-organizing
 - system, 20
- servent, 24
- home, 19
- SHA-1, 31
- similarity, 1, 2
 - algebra with weights, 6
 - approximate search, 12
 - query, 3–7
 - search, xiv, 1–3
- Similarity Hashing, 13
- Simple Algorithm, 50
- Simple Bloom Petal Algorithm, 50
- SIP, 75
- small-world
 - access methods, 51
 - network, 51
 - phenomenon, 51
- space-filling curves, 39
- Spaghettis, 13, 81
- Spatial Approximation Tree, 13
- spatial index, 52
- strict contraction, *see* mapping, Strict contraction
- superpeers, 24
- surrogate routing, 42
- SWAM-V, 51
- Symphony, 38, 46–47
- takeover node, 69
- Tapestry, 41–42, 74
- threshold algorithm, 50
- Time to Live, 28
- Topology-Aware Routing, 49
- tree edit distance, *see* distance, tree edit
- tree-based application level multicast, 74
- TTL, *see* Time to Live

- Vantage Point Tree, *see* VPT
- Viceroy, 38, 47–49
- VID, *see* virtual identifier, *see* virtual identified65
- virtual identifier, 65
- VoIP, 75
- Voronoi, 52
- VPT, 12, 54
- VPT*, 53–57

- XML, 11
- XOR metric, 44

- Yahoo
 - Image Search, xv

Z-Ring, 42–43

zone, 63

 adjacent, 63