On Scalability of the Similarity Search in the World of Peers

Michal Batko
Masaryk University
Brno, Czech Republic

David Novak
Masaryk University
Brno, Czech Republic

Fabrizio Falchi
ISTI-CNR
Pisa, Italy

Pavel Zezula
Masaryk University
Brno, Czech Republic

Abstract

Due to the increasing complexity of current digital data, similarity search has become a fundamental computational task in many applications. Unfortunately, its costs are still high and the linear scalability of single server implementations prevents from efficient searching in large data volumes. In this paper, we shortly describe four recent scalable distributed similarity search techniques and study their performance of executing queries on three different datasets. Though all the methods employ parallelism to speed up query execution, different advantages for different objectives have been identified by experiments. The reported results can be exploited for choosing the best implementations for specific applications. They can also be used for designing new and better indexing structures in the future.

1. Introduction

Efficient lookup for specific keywords in a dictionary of more than 100,000 words or locating specific records in millions of bank accounts are quite easy tasks for present-day computers. Since records in such domains can be sorted, and every record either fully satisfies the search condition or it does not at all, hashing or tree-like structures can be applied as indexes. High scalability of such technologies is guaranteed by logarithmically bounded search time with respect to the size of the file.

However, to find images of sport cars, time series with similar development, or groups of customers with common buying patterns in respective data collections, the traditional technologies simply fail. Here, 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 certain relation measured by a user defined dissimilarity function. The importance of such a search is increasing for a variety of present complex digital data collections and is generally designated as the *similarity search*.

Although many similarity search approaches have been proposed, the most generic one considers the mathematical *metric space* as a suitable abstraction of similarity [27]. The simple but powerful concept of the metric space consists of a *domain* of objects and a *distance function* that measures proximity of pairs of objects. 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 nature, etc. Despite many index structures have been proposed, the similarity search is inherently expensive. Besides, the linear scalability of the search time prevents from application to huge files that have become common with a prediction of continuous rapid growth.

Very recently, we have proposed four scalable and distributed similarity search structures for metric data. The first two structures respect the basic *ball* and *generalized hyperplane* partitioning principles [26] and they are called the *VPT** and the *GHT**, respectively [4]. The other two apply transformation strategies the metric similarity sear ch problem is transformed into a series of range queries executed on existing distributed keyword structures, namely the *CAN* [21] and the *Chord* [24]. By analogy, we call them the *MCAN* [10] and the *M-Chord* [19]. Each of the structures is able to execute similarity queries for any metric dataset, and they all exploit parallelism for query execution. However, due to the completely different underlying principles, an important question arises: What are the advantages and disadvantages of the individual approaches in terms of search costs and scalability for different real-life search problems?

In this paper, 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 the most important findings. We focus on scalability with respect to the size of the query, the size of the dataset, and the number of simultaneously executed queries.

The rest of the paper is organized as follows. The necessary background and related work are reported in Section 2. A brief specification of our four indexing techniques is available in Section 3, while the assumptions and results of our experiments can be found in Section 4. The paper concludes in Section 5.

INFOSCALE '06. Proceedings of the First International Conference on Scalable Information Systems, May 29-June 1 2006, Hong Kong

2. Background and related work

In this section, we provide a theoretical background for the similarity search. Then we mention the distributed paradigm adopted by all presented solutions. We also give a brief related work survey in the end of this section.

2.1. Metric-based similarity search

Let us recall basic definitions and principles that are necessary for the metric-based indexing.

Definition 1: *Metric space* $\mathcal{M}$ is a pair $\mathcal{M} = (\mathcal{D}, d)$, where $\mathcal{D}$ is the *domain* of objects and d is the total *distance function* $d : \mathcal{D} \times \mathcal{D} \longrightarrow \mathbb{R}$ satisfying the following conditions for all objects $x, y, z \in \mathcal{D}$:

$d(x,y) \geq 0$	(*non-negativity*),
$d(x,y) = 0$ iff $x = y$	(*identity*),
$d(x,y) = d(y,x)$	(*symmetry*),
$d(x,z) \leq d(x,y) + d(y,z)$	(*triangle inequality*).

Several types of similarity queries are defined in the literature, but we concentrate on the two most common ones the *range query* and the *k-nearest neighbors query*. Let $I \subseteq \mathcal{D}$ be a finite set of data objects indexed by an index structure.

Definition 2: Given an object $q \in \mathcal{D}$ and a maximal search radius r, *range query* $\mathbf{Range}(q, r)$ selects a set $S_A \subseteq I$ of indexed objects: $S_A = \{x \in I \,|\, d(q,x) \leq r\}$.

Definition 3: Given an object $q \in \mathcal{D}$ and an integer $k \geq 1$, *k-nearest neighbors query* $\mathbf{kNN}(q, k)$ retrieves a set $S_A \subseteq I : |S_A| = k, \forall x \in S_A, \forall y \in I \setminus S_A : d(q,x) \leq d(q,y)$.

Since there is no coordinate system in the metric world, the only way to divide and prune the indexed data space is to use relative distances from some preset objects.

The following principle based on the triangle inequality of d is behind most of the metric index methods. For a $\mathbf{Range}(q, r)$ query, every indexed object $x \in I$ may be excluded without evaluating $d(q, x)$ if

$$|d(p,x) - d(p,q)| > r \tag{1}$$

where $d(q,p)$ and $d(x,p)$ are precomputed distances to some fixed object $p \in \mathcal{D}$. Furthermore, having a set of n objects (*pivots*) $p_0, \ldots, p_{n-1} \in \mathcal{D}$ and having values of $d(x,p_0), \ldots, d(x,p_{n-1})$ for an object $x \in \mathcal{D}$, this object can be excluded if

$$\exists i, 0 \leq i < n : |d(p_i,x) - d(p_i,q)| > r. \tag{2}$$

In the following, we refer to this formula as the *pivot filtering criterion* [27].

2.2. Peer-to-peer paradigm

All the presented systems are based on the P2P philosophy and constitute purely decentralized structured P2P networks. These features refer to the fact that *peers* (nodes participating in the network) offer the same functionality and the system follows some distributed logic that facilitates an effective intra-system navigation.

Generally, every node of such a system provides the following features and requires them from the other peers:

- *resources* storage and computational power,
- *communication* every node can contact any other node directly if knowing its network identification,
- *navigation* internal structure that ensures correct routing among the peers.

To ensure a maximal scalability, all the systems also adopt the requirements of the *Scalable and Distributed Data Structures* [17]:

- data expands to new nodes gracefully, and only when the nodes already used are efficiently loaded;
- there is no master site to be accessed when searching for objects, e.g., there is no centralized directory;
- the data access and maintenance primitives, e.g., search, insertion, split, etc., never require atomic updates to multiple nodes.

2.3. Related work

Many metric-based indexing principles and index structures have been proposed, focusing on pruning of the search space at query time [12, 8, 27]. However, even with the most sophisticated techniques, the similarity search becomes too expensive when the stored data volume grows, because the search costs increase linearly with respect to the size of the dataset [9]. This fact calls for an attempt to exploit a distributed processing.

Restricting to multi-dimensional interval queries in *vector spaces*, several distributed structures have been proposed recently, for example the MAAN [7], MURK [11], Mercury [5] or Skip Graphs [1]. A general solution for the range and nearest neighbors search in the vector data is provided in the SWAM [2] a family of small-world based access methods. Unfortunately, these structures are often designed for specific applications (for example spatial data) typically using vectors of low dimensionality. The vector space approach cannot be applied on many important datasets where similarities are measured by functions such as the Hausdorff distance, Jaccard's coefficient, edit distance, etc.

On the other hand, the pSearch [25] introduces a decentralized P2P information retrieval system based on the CAN routing protocol. However, this approach is only suitable for the text retrieval. To the best of our knowledge, the four systems elaborated in this paper are the only published metric-based distributed data structures. In this respect, this paper presents the first and extensive performance comparison of all distributed metric similarity search indexes.

3. Distributed metric approaches

This section contains short descriptions of four different distributed structures for indexing and similarity search in the metric data. The first two, *GHT** and *VPT**, are native metric index structures whereas the other two, *MCAN* and *M-Chord*, transform the metric search issue into a different problem and take advantage of some well-known solutions. Each description consists of a main idea of the particular approach, a basic architecture of the system, and a schema of algorithms for the **Range** queries. All the structures adopt very similar approach to solve the **kNN** queries and this technique is explained in the end of this section.

3.1. GHT* & VPT*

In this section, we describe two distributed metric index structures the *GHT** [4] and its not yet published extension called the *VPT**. Both of them exploit natural metric partitioning principles that are used to build a distributed binary tree [15].

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 store objects. It may be, for example, a memory segment or a block on a disk. The number of buckets managed by a peer depends on its own potential.

Since both the 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 different peer than the original one. Thus, the structures grow as new data come in.

The core of the algorithm lays down a mechanism for locating appropriate peers which hold requested objects. The part of the structure responsible for this navigation is called the *Address Search Tree* (AST). In order to avoid hotspots, which may be caused by the existence of a centralized node accessed by every request, an instance of the AST structure is present in 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 in the peers contain only limited navigation information which may be imprecise. The locating step is then repeated on contacted peers until the desired peers are reached. The algorithm guarantees that the destination peers are always found. Both the structures also provide a mechanism called *image adjustment* for updating the imprecise parts of the AST automatically. We will focus only on the basics of both the structures, i.e. the partitioning principles used in AST and the evaluation of range queries. For more details see [4].

Address search tree The AST is a binary search tree based on the Generalized Hyperplane Tree (GHT) [26] in *GHT**, and on the Vantage Point Tree (VPT) [26] 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 bucket (denoted as BID) or a peer (denoted as NNID) holding the data. Whenever the data is in a bucket on the local peer, a leaf node is a BID pointer. An NNID pointer is used if the data is on a remote peer.

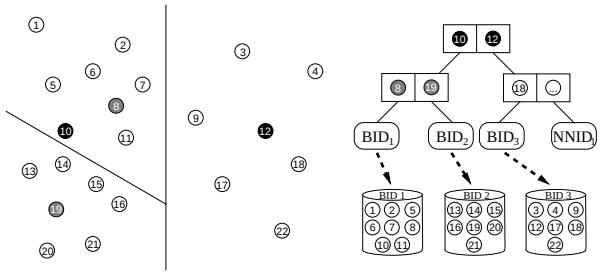

Figure 1. Address Search Tree with the generalized hyperplane partitioning

An example of AST using the generalized hyperplane partitioning is depicted in Figure 1. In order to divide a set of objects $I = \{o_1, \ldots, o_{22}\}$ 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 1, we select objects o_{10}, o_{12} and call them *pivots* for the first level of the AST. Then, the original set I is split by measuring the distance between every object $o \in I$ and both the pivots. If $d(o_{10}, o) \leq d(o_{12}, o)$, i.e. the object o is closer to the pivot o_{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 build accordingly. Figure 1 shows an example of such a tree. Observe that the leaf nodes are denoted by BID_i and $NNID_i$ symbols. That means that the corresponding parti-

tion (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. In general, this principle also allows to divide a set I into two partitions I_1 and I_2. However, only one pivot o_{11} is selected from the set and the objects are divided by a radius r_1. More specifically, if the distance between the pivot o_{11} and an object $o \in I$ is smaller or equal to the specified radius r_1, i.e. if $d(o_{11}, o) \leq r_1$ then the object belongs to partition I_1. The object is assigned to I_2 otherwise. Similarly, the algorithm is used recursively to build a binary tree. The leaf nodes also follow the same schema for addressing local buckets and remote peers.

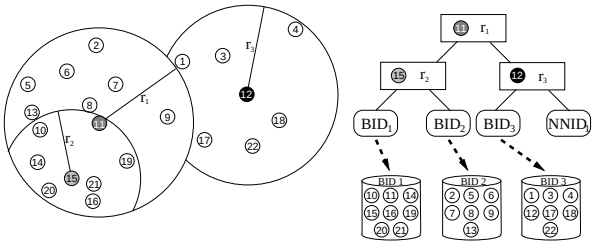

Figure 2. Address Search Tree with the vantage point partitioning

Range search The search for $\mathbf{Range}(q, r)$ query 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 and thus the inner node format $\langle p_1, p_2 \rangle$:

$$d(p_1, q) - r \leq d(p_2, q) + r \tag{3}$$

$$d(p_1, q) + r > d(p_2, q) - r \tag{4}$$

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

$$d(p, q) - r \leq r_p \tag{5}$$

$$d(p, q) + r > r_p \tag{6}$$

The right subtree of the inner node is traversed if Condition 3 for the *GHT** or Condition 5 for the *VPT** qualifies. The left subtree is traversed whenever Condition 4 or Condition 6 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 recursively 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 using Equation 1. All qualifying objects form the query response set.

3.2. MCAN

In order to manage metric data, the *MCAN* [10] uses a pivot-based technique that maps data objects $x \in \mathcal{D}$ to an N-dimensional vector space $\mathbb{R}^N$. Then, the *CAN* [21] Peer-to-Peer protocol is used for partitioning the space and the internal navigation. Having a set of N pivots $p_1, \ldots, p_N$ selected from $\mathcal{D}$, *MCAN* maps an object $x \in \mathcal{D}$ to the vector space by means of the following function $F : \mathcal{D} \to \mathbb{R}^N$:

$$F(x) = (d(x, p_1), d(x, p_2), \ldots, d(x, p_N)). \tag{7}$$

The virtual vector space coordinates designate the object x placement within the *MCAN* structure. The *CAN* protocol divides the vector space into regions and assigns them to the participating peers. The object x is stored by the peer whose region contains $F(x)$. Using L^∞ as a distance function in the vector space, the mapping F is *contractive*, i.e. $L^\infty(F(x), F(y)) \leq d(x, y)$. It can be proved using the triangle inequality of the metric function d[10]. Thus, the algorithm for $\mathbf{Range}(q, r)$ query involve only the regions that cover objects x for which $L^\infty(F(x), F(q)) \leq r$. In other words, it accesses the regions that intersect the hypercube with side $2r$ centered in $F(q)$ (see Figure 3).

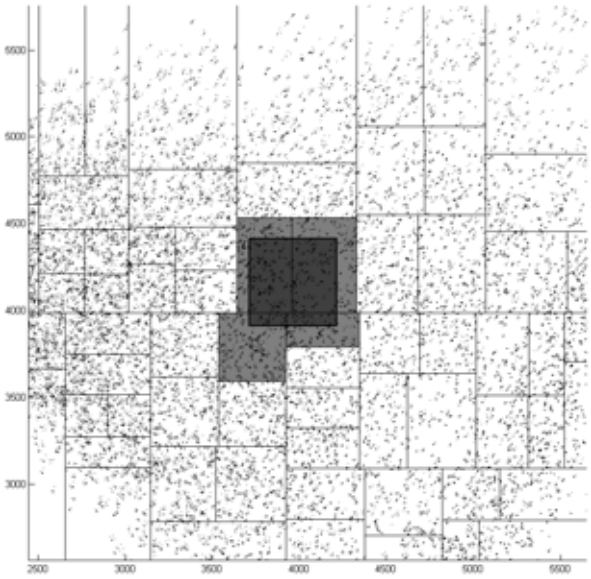

Figure 3. Example of MCAN range query

In order to further reduce the number of evaluated distances, *MCAN* uses the additional pivot-based filtering according to Formula 2. All peers use the same set of pivots: the N pivots from the mapping function F (Equation 7), plus additional pivots since N is typically low. All the pivots are selected from a sample dataset using the incremental selection technique [6].

Routing in *MCAN* works in the same way as for the original *CAN*. Every peer maintains a coordinate-based routing table containing the network identifiers and the coordinates of its neighboring peers in the virtual $\mathbb{R}^N$ space. In every step, the routing algorithm passes the query to the neighboring peer that is geometrically the closest to the target point in the space. Given a dataset, the average number of neighbors per peer is proportional to the dimensionality N while the average number of hops to reach a peer is inversely proportional to this value [21].

The insert operation When inserting an object $x \in \mathcal{D}$ into *MCAN*, the initiating peer computes distances between x and all pivots. These values are used for mapping x into $\mathbb{R}^N$ by Equation 7 and then the insertion request is forwarded (using the *CAN* navigation) to the peer that covers value $F(x)$. The receiving peer stores x and if it reaches its storage capacity limit (or another defined condition) it executes a split. The peer's region is split into two parts trying to divide the storage equally. One of the new regions is assigned to the new active peer and the other one replaces the original region.

Range search algorithm The peer that initiates a $\mathbf{Range}(q, r)$ query first computes distances between q and all the pivots. The *CAN* protocol is then employed in order to reach the region which covers $F(q)$. If a peer, visited during the routing process, intersects the query area, the request is spread to all other involved peers using a multicast algorithm described in detail in [14, 22]. Every affected peer searches its data storage employing the pivot filtering mechanism and returns the answer directly to the initiator.

3.3. M-Chord

Similarly to the previous *MCAN* method, the *M-Chord* [19] approach also transforms the original metric space. The core idea is to map the data space into a one-dimensional domain and use this domain together with the *Chord* routing protocol [24].

In particular, this approach exploits the idea of a vector index method *iDistance* [13] 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

$$idist(x) = d(p_i, x) + i \cdot c.$$

Figure 4a visualizes the mapping schema. Handling a $\mathbf{Range}(q, r)$ query, the space to be searched is specified by *iDistance* intervals for such clusters that intersect the query sphere see an example in Figure 4b.

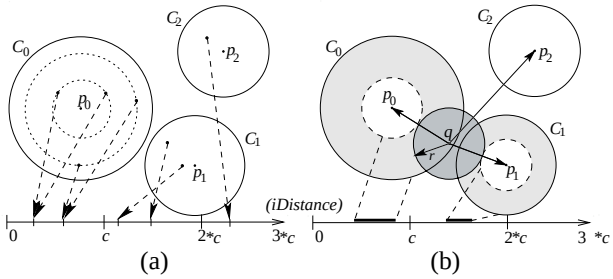

Figure 4. The principles of iDistance

This method is generalized to metric spaces in the *M-Chord*. No coordinate system can be used to partition a general metric space, therefore, a set of n pivots $p_0, \ldots, p_{n-1}$ is selected from a sample dataset and the space is partitioned according to these pivots. The partitioning is done in a Voronoi-like manner [12] (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 a uniform order-preserving hash function h into the *M-Chord* domain of size 2^m. Thus, for an object $x \in C_i$, $0 \leq i < n$, the *M-Chord* key-assignment formula becomes:

$$\textit{m-chord}(x) = h(d(p_i, x) + i \cdot c) \qquad (8)$$

The M-Chord structure Having the data space mapped into the one-dimensional *M-Chord* domain, every active node of the system takes over responsibility for an interval of keys. The structure of the system is formed by the *Chord* circle [24]. This Peer-to-Peer protocol provides an efficient localization of the node responsible for a given search key.

When inserting an object $x \in \mathcal{D}$ into the structure, the initiating node N_{ins} computes the *m-chord*(x) key through Formula 8 and employs the *Chord* to forward a store request to the node responsible for the computed key (see Figure 5a).

The nodes store the data in a B^+-tree storage according to their *M-Chord* keys. When a node reaches its storage capacity limit (or another defined condition) it executes 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 $\mathbf{Range}(q, r)$ 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 then spreads to all nodes covering the particular interval (see Figure 5b).

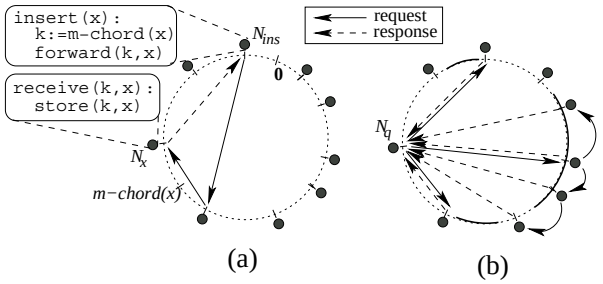

Figure 5. The insert (a) and range search (b)

The *iDistance* pruning technique filters out all objects $x \in C_i$ that fulfil $|d(x, p_i) - d(q, p_i)| > r$. When inserting an object x into *M-Chord*, distances $d(x, p_i)$ are computed $\forall i : 0 \leq i < n$. These values are stored together with object x and the general metric filtering criterion (Equation 2) improves the pruning of the search space.

3.4. Nearest-neighbors search

The previous brief descriptions of the structures do not mention algorithms for $\mathbf{kNN}$ queries. Generally, all the systems adopt a similar approach of $\mathbf{kNN}(q, k)$ query evaluation, which exploits the range search. The idea is to estimate radius r, so that the $\mathbf{Range}(q, r)$ query returns at least k objects.

More precisely, the general $\mathbf{kNN}$ algorithm has the following two phases:

1. Send a request to the node where object q would be stored and search for k objects that are near q. Measure the distance r to the k^{th} nearest object found.
2. Execute the $\mathbf{Range}(q, r)$ query and return the k nearest objects from the query result (skip the space searched during the first phase).

If less than k objects are found in the storage during the first phase then some other estimation techniques are used see [3] for details.

The space limitations do not permit us to present $\mathbf{kNN}$ performance results and, thus, the following section, that evaluates the performance of the structures, concerns the $\mathbf{Range}$ search only. Because the $\mathbf{kNN}$ algorithm directly exploits the $\mathbf{Range}$ search, the scalability trends are very similar for both algorithms and all the presented results are relevant for the $\mathbf{kNN}$ query processing as well.

4. Scalability performance evaluation

In this section, we provide comparison of the presented approaches by confronting results of 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. Due to these facts, we consider the results of the experiments fairly comparable.

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

4.1. Experiments settings

All the compared systems are dynamic. Each structure maintains a set of available inactive nodes and employs these to split the overloaded nodes, although other splitting scenarios are possible as well. 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 storage capacity of 5,000 objects. The implementations built up 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:

VEC 45-dimensional *vectors* of extracted color image features. The similarity function d for comparing the vectors is a *quadratic-form distance* [23]. 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* are of lengths from 3 to 200 characters and are compared by the *edit distance* [16] on the level of individual characters. The distance distribution of this dataset is skewed.

DNA protein symbol *sequences* of length sixteen. The sequences are compared by a *weighted edit distance* according to the Needleman-Wunsch algorithm [18]. This distance function has a very 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, the whole datasets of 1,000,000 objects are used (900,000 for **TTL**). As for other settings that are specific for 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 number of pivots according to the depth of the AST tree (see Section 3.1).

All the presented performance characteristics of query processing have been taken as an average over 100 queries with randomly chosen query objects.

4.2. Measurements

In real applications as well as in the described datasets, evaluation of the distance function d has typically 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 the structure efficiency. The CPU costs of other operations (and usually I/O cost 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 a query:

- *total distance computations* the sum of the number of the distance function evaluations on all engaged peers,
- *parallel distance computations* the maximal number of distance evaluations performed in a sequential manner during the 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. Instead, we use the parallel distance computations together with the maximal hop count as an objective response time estimation. Another indicator that we monitored is the *percentage of nodes* that were affected by the query processing.

4.3. Changing the query size

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 $\mathbf{Range}(q, r)$ 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% which resulted in approximately 150 active nodes for each tested system.

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) and 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.

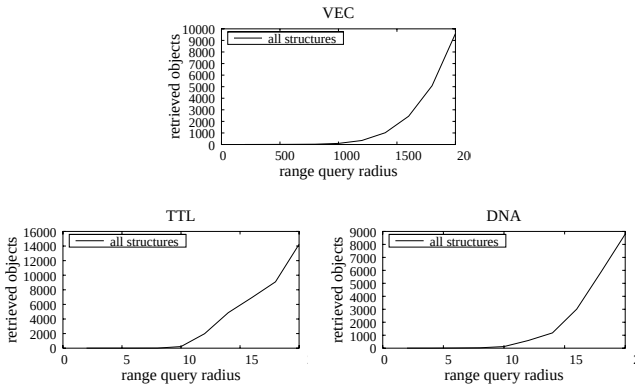

Figure 6. Number of retrieved objects

In the first group of graphs, shown in Figure 6, we report on the relation between the query radius size and the number of retrieved objects. As intuitively clear, the bigger 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 the three datasets. Note that, for example, the biggest radius 2,000 in the **VEC** dataset selects almost 10,000 objects (2% of the whole database), for the **TTL** dataset the biggest radius retrieves even more objects. Obviously, such big radii are usually not reasonable for applications (e.g., two titles with edit distance 20 differ a lot), but we provide the results in order to study behavior of the structures also in these cases. On the other hand, smaller radii return reasonable amounts of objects, for instance, radius 6 results in approximately 30 objects in the **DNA** dataset, which is not clearly readable from the graphs.

The number of visited nodes is reported in Figure 7. 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 struc-

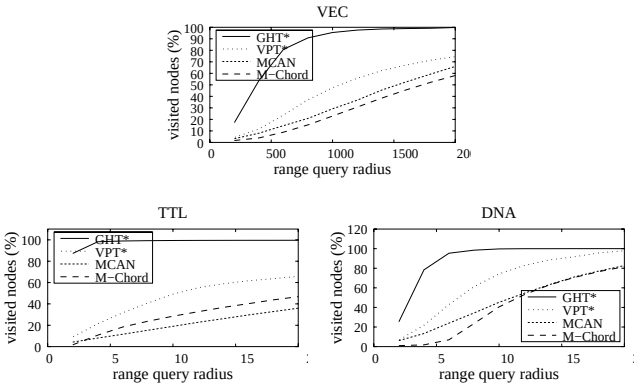

Figure 7. Percentage of visited nodes

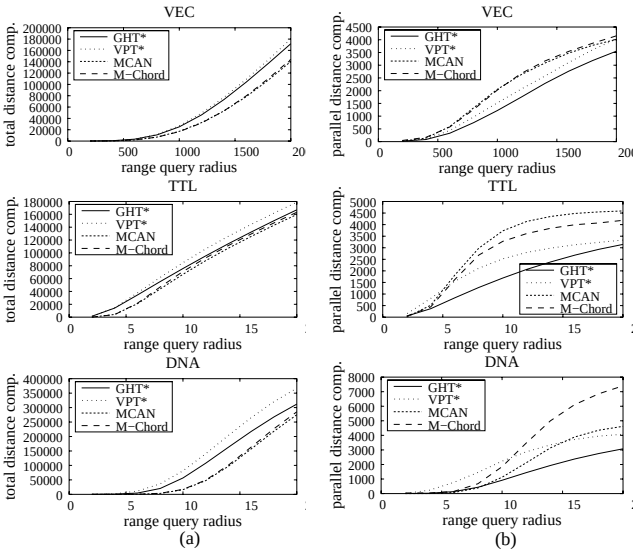

Figure 8. The total (a) and parallel (b) number of distance computations

ture. As mentioned earlier, the total number of active peers participating in the network was around 150, thus, value 20% in the graph means that approximately 30 peers were used to complete the results. 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 all the participating nodes very soon as the radius grows. This is induced by the fact that the generalized hyperplane partitioning does not guarantee a balanced split as opposed to the other three methods. Moreover, because we count all the nodes that evaluate distances as visited, the *VPT** and the *GHT** algorithms are a little bit handicapped. Recall that they need to compute distances to pivots during the navigation and thus the nodes that only forwards the query are also counted as visited.

Note that the used 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 by Figure 8, shows the computational costs with respect to the query radius. We provide a pair of graphs for every dataset. The graphs on the left (a) report the total number of distance computations needed to evaluate a range query. This measure can be considered to be the query costs in centralized index structures. The graphs on the right (b) 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 (as mentioned in Section 2) to avoid as much distance computations as possible. 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 than for 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 filtered using maximally 18 and minimally 5 pivots.

Observe the effects of filtering on total distance computations in Figure 8a. We can see that the *M-Chord* and *MCAN* structures have practically the same filtering for all the datasets. On the other hand, the *VPT** index is always the worst, since it has the lowest number of pivots used for filtering. We can also 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 as for the **TTL** and **VEC** datasets. Then, queries with bigger radii in the **DNA** dataset have to access about 60% of the whole database, which would be very slow in a centralized index.

Figure 8b illustrates the parallel computational costs of the query processing. We can see that the amount of necessary distance computations is significantly reduced, which comes out 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 7), which in fact is strictly 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 is best visible 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 into 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 could be exceeded.

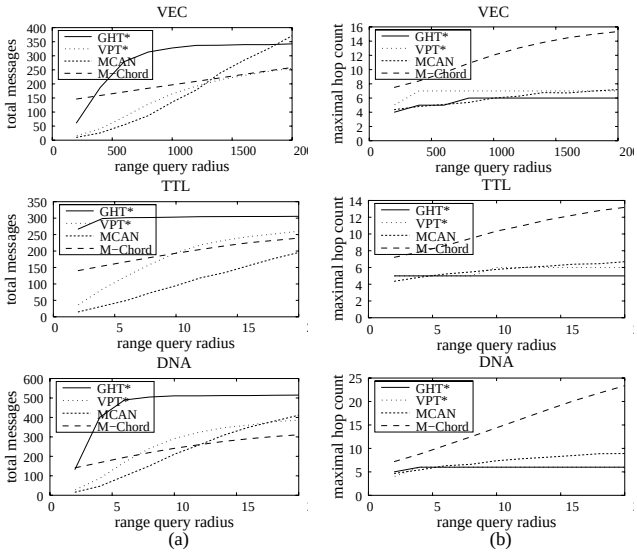

Figure 9. The total number of messages (a) and the maximal hop count (b)

The last group of measurements in this section, depicted in Figure 9, reports on the communication costs, i.e. the traffic load of the underlying network. Since all the structures exploit the message passing paradigm and the amount of interchanged data is small (usually fitting in a few packets), we measure the number of messages needed to solve a range query as the communication cost. By analogy, we show the total messages cost, which can be interpreted as the overall load of the underlying network infrastructure, and the parallel cost represented by the maximal hop count . Recall that the hop count is the number of messages sent in a serial manner, i.e. the sequence of messages forwarded from one node to another.

Since the *GHT** and the *VPT** count all nodes involved in navigation as visited (as explained earlier), the percentage of visited nodes (Figure 7) and the total number of messages (Figure 9a) 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 specific query request several times. However, every peer evaluates a particular 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. In fact, the *M-Chord* needs to access at least one peer for every *M-Chord* cluster even for small range queries, see Section 3.3. Then, as the radius increases, adjacent peers within some of the clusters need to be contacted and that increases the total messages costs.

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

4.4. Increasing the dataset size

Let us concern the systems' scalability with respect to the growing volume of data stored in the structures. We have observed the performance of $\mathbf{Range}(q, r)$ queries on systems storing from 50,000 to 1,000,000 objects. We conducted these experiments on all datasets 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.

The space limitations do not permit us to present all the collected results, therefore, we include only one graph for each type of measurement if the other graphs exhibit the same trend. The title of each graph in this section specifies the used dataset and the search radius r.

The number of retrieved objects see, e.g., the radius 10 for the **TTL** dataset in Figure 10a grows precisely linearly because the data were inserted to the structures in random order.

Figure 10b 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

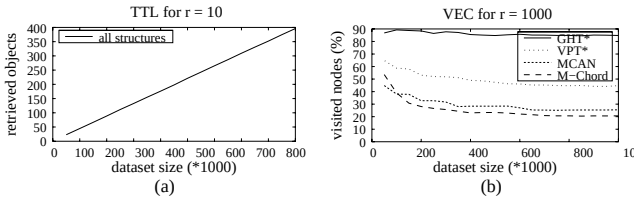

Figure 10. Retrieved objects (a) and percentage of visited nodes (b) for growing dataset

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 4.3, the *GHT** partitioning is not balanced, therefore, the query processing is spread over larger number of participating nodes.

Figure 11 presents the computational costs in terms of both total and parallel number of distance computations. As expected, the total costs (a) increase linearly with the stored data volume. 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, as explained in Section 4.3. The total costs for the *GHT** and the *VPT** are slightly higher due to the dynamic sets of filter pivots.

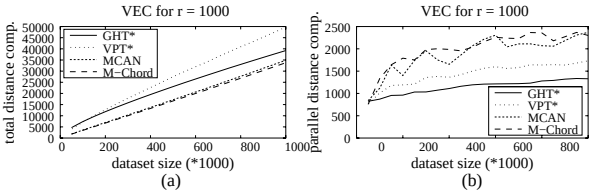

Figure 11. The total (a) and parallel (b) number of distance computations

The parallel number of distance computations (Figure 11b) 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% increase for doubled dataset size from 500,000 to 1,000,000. The increase is caused by the fact that the involved nodes contain more of the relevant objects while making the data space denser. This corresponds with the observable correlation of this graph and Figure 10b the less nodes the structure involves, the higher the parallel costs it shows. The transformation techniques, the *MCAN* and the *M-Chord*, obviously concentrate the relevant data on fewer nodes and have higher parallel costs then. The noticeable graph fluctuations are caused by rather regular splits of overloaded nodes.

Figure 12 presents the same results for **DNA** dataset. The pivot-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 relatively small variety of possible values. As mentioned in Section 4.3, for this dataset, the *M-Chord* mapping collisions may result in overloaded nodes that cannot be split. Then, the parallel costs in Figure 12b may be over the split limit of 5,000 objects.

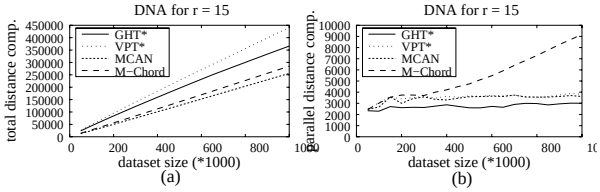

Figure 12. The total (a) and parallel (b) number of distance computations

Figure 13 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 but the major increase of the messages sending is in sequential manner which negatively influences the hop count.

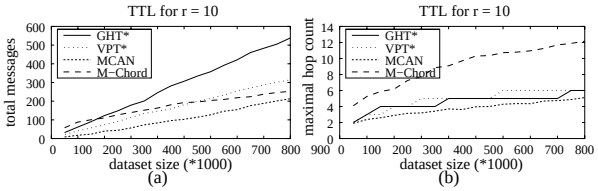

Figure 13. The total messages (a) and the maximal hop count (b)

4.5. Number of simultaneous queries

In this section, we focus on scalability of the systems with respect to the number of queries executed simultaneously. In other words, we measure the *interquery* parallelism [20] of the queries processing.

In the conducted experiments, we have simultaneously executed groups of 10 100 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 inter-node 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 this

experiments for all datasets and have used the same query radii as in Section 4.4.

In order to establish a baseline, we have measured 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 also be interpreted as the number of queries that can be handled by the systems simultaneously without slowing them down.

Looking at Figures 14a, 15a and 16a, 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.

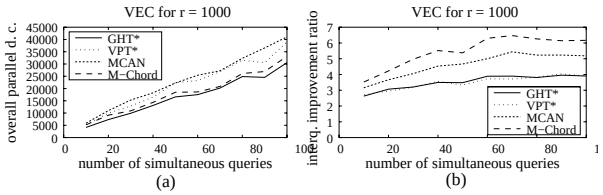

Figure 14. The overall parallel costs (a) and interquery improvement ratio (b)

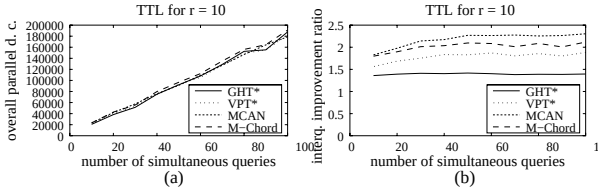

Figure 15. The overall parallel costs (a) and interquery improvement ratio (b)

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 better than *GHT**.

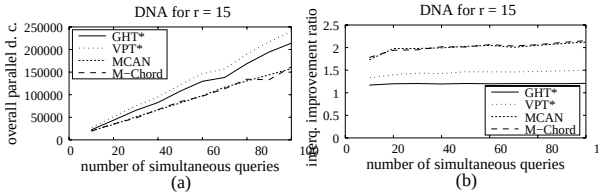

Figure 16. The overall parallel costs (a) and interquery improvement ratio (b)

The actual improvement ratio values for specific datasets are strongly influenced by the total number of distance computations spread over the nodes (see Figure 8a) and, therefore, the improvement is lower for **DNA** than for **VEC**.

5. Conclusions

In this paper, we have studied performance of four different distributed metric index structures, 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) increasing query radii, (2) growing volume of searched data, and (3) accumulating number of concurrent queries. We have conducted a vast bulk of experiments and reported the most interesting findings in a special section of this paper.

Though 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.

	single query	*multiple queries*
*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 the execution of one 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 considered 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 solved by the *GHT** are practically executed 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 a characteristic subset of the indexed data 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, unfortunately it does, for example due to the luck of the characteristic subset, 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 reviled 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, that is designing structures with respect to the user defined bounds on the query response time.

References

[1] J. Aspnes and G. Shah. Skip graphs. In *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384 393, January 2003.

[2] F. Banaei-Kashani and C. Shahabi. SWAM: A family of access methods for similarity-search in peer-to-peer data networks. In *CIKM '04: Proceedings of the Thirteenth ACM conference on Information and knowledge management*, pages 304 313. ACM Press, 2004.

[3] M. Batko, C. Gennaro, and P. Zezula. A scalable nearest neighbor search in p2p systems. In *Procedings of DBISP2P*, volume 3367 of *Lecture Notes in Computer Science*, pages 79 92, 2004.

[4] M. Batko, C. Gennaro, and P. Zezula. Similarity grid for searching in metric spaces. In *DELOS Workshop: Digital Library Architectures, Lecture Notes in Computer Science*, volume 3664/2005, pages 25 44, 2005.

[5] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. *SIGCOMM Comput. Commun. Rev.*, 34(4):353 366, 2004.

[6] B. Bustos, G. Navarro, and E. Chvez. Pivot selection techniques for proximity searching in metric spaces. In *Proc. of SCCC01*, pages 33 40, 2001.

[7] M. Cai, M. Frank, J. Chen, and P. Szekely. MAAN: A multi-attribute addressable network for grid information services. In *GRID '03: Proceedings of the Fourth International Workshop on Grid Computing*, pages 184 191, Washington, DC, USA, 2003. IEEE Computer Society.

[8] E. Chavez, G. Navarro, R. Baeza-Yates, and J. L. Marroquın. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273 321, 2001.

[9] V. Dohnal, C. Gennaro, P. Savino, and P. Zezula. D-Index: Distance searching index for metric data sets. *Multimedia Tools and Applications*, 21(1):9 33, 2003.

[10] F. Falchi, C. Gennaro, and P. Zezula. A content-addressable network for similarity search in metric spaces. In *Proceedings of DBISP2P*, pages 126 137, 2005.

[11] P. Ganesan, B. Yang, and H. 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.

[12] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst.*, 28(4):517 580, 2003.

[13] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. iDistance: An adaptive b+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30(2):364 397, 2005.

[14] M. B. Jones, M. Theimer, H. Wang, and A. Wolman. Unexpected complexity: Experiences tuning and extending can. Technical Report MSR-TR-2002-118, Microsoft Research, December 2002.

[15] B. Kr¨oll and P. Widmayer. Distributing a search tree among a growing number of processors. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data / SIGMOD '94, Minneapolis, Minnesota*, pages 265 276, 1994.

[16] V. I. Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission*, 1:8 17, 1965.

[17] W. Litwin, M.-A. Neimat, and D. A. Schneider. LH* A scalable, distributed data structure. *ACM Transactions on Database Systems*, 21(4):480 525, 1996.

[18] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal Of Molecular Biology*, 48:443 453, 1970.

[19] D. Novak and P. Zezula. M-Chord: A scalable distributed similarity search structure. In *Proceedings of First International Conference on Scalable Information Systems (INFOSCALE 2006), Hong Kong, May 30 June 1* . IEEE Computer Society, 2006.

[20] M. T. Özsu and P. Valduriez. Distributed and parallel database systems. *ACM Comput. Surv.*, 28(1):125 128, 1996.

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

[22] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. *Lecture Notes in Computer Science*, 2001.

[23] T. Seidl and H.-P. Kriegel. Efficient user-adaptable similarity search in large multimedia databases. In *The VLDB Journal*, pages 506 515, 1997.

[24] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, pages 149 160. ACM Press, 2001.

[25] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks, 2002.

[26] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175 179, 1991.

[27] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer-Verlag, 2006.