Available online at www.sciencedirect.com
ScienceDirect
Future Generation Computer Systems 24 (2008) 834–848
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

Scalability comparison of Peer-to-Peer similarity search structures

Michal Batko[a,*], David Novak[a], Fabrizio Falchi[b], Pavel Zezula[a]

[a] *Masaryk University, Brno, Czech Republic*
[b] *ISTI-CNR, Pisa, Italy*

Received 28 January 2007; received in revised form 20 July 2007; accepted 29 July 2007
Available online 6 August 2007

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 grow linearly on single server structures, which prevents them from efficient application on large data volumes. In this paper, we shortly describe four recent scalable distributed techniques for similarity search and study their performance in 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 would be helpful for choosing the best implementations for specific applications. They can also be used for designing new and better indexing structures in the future.

Keywords: Similarity search; Scalability; Metric space; Distributed index structures; Peer-to-Peer networks

1. Introduction

Efficient lookup for specific keywords in a dictionary containing hundreds of thousands of 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. Such a searching is more and more important for a variety of current 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 [1]. 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. Although many index structures have been proposed, the similarity search is inherently expensive. Besides, the linear increase of the search time prevents them from application to huge files that have become common and the predictions expect a continuous rapid growth.

Very recently, we have proposed four scalable distributed similarity search structures for metric data. The first two structures adopt the basic *ball* and *generalized hyperplane* partitioning principles [2] and they are called the *VPT** and the *GHT**, respectively [3]. 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* [4] and the *Chord* [5]. By analogy, we call them the *MCAN* [6] and the *M-Chord* [7]. 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

* Corresponding address: Masaryk University, Faculty of Informatics, Botanicka 68a, 60200 Brno, Czech Republic. Tel.: +420 549493232.
E-mail addresses: xbatko@fi.muni.cz (M. Batko), xnovak8@fi.muni.cz (D. Novak), fabrizio.falchi@isti.cnr.it (F. Falchi), zezula@fi.muni.cz (P. Zezula).

0167-739X/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2007.07.012

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 queries executed simultaneously. The results reported in this paper have been presented on the INFOSCALE '06 conference [8].

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.

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 focus 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 subset $S_A \subseteq I$ of indexed objects such that $S_A = \{x \in I \mid d(q, x) \leq r\}$.

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 subset $S_A \subseteq I$ such that $|S_A| = k, \forall x \in S_A, \forall y \in I \setminus S_A : d(q, x) \leq d(q, y)$.

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

The following principle based on the properties of d is behind most of the metric indexing methods. For a $\mathbf{Range}(q, r)$ query, every indexed object $x \in I$ may be excluded from the result without evaluating $d(q, x)$ if $|d(x, p) - d(q, p)| > r$, where $d(x, p)$ and $d(q, p)$ 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

$$\exists i, \quad 0 \leq i < n : |d(p_i, x) - d(p_i, q)| > r. \tag{1}$$

In the following, we refer to this formula as the *pivot-filtering criterion* [1].

2.2. Peer-to-Peer paradigm

All the presented systems are based on the P2P philosophy and constitute purely-decentralized structured P2P networks. These terms 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 consists of the following components and expects 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* [9]:

- 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 the search space at query time [10,11,1]. However, even with the most sophisticated techniques, the similarity search becomes too expensive when the volume of stored data grows, because the search costs increase linearly with respect to the dataset size [12]. This fact calls for an attempt to exploit a distributed processing.

After the boom of Distributed Hash Tables, the decentralized structured P2P networks began to focus on efficient processing of single-dimensional interval queries — see, for example, P-Grid [13], Skip Graphs [14], SkipNet [15] or P-Ring [16]. Recently, the P2P query-paradigm has been generalized to multi-dimensional interval queries in vector spaces which is supported, for instance, by the following structures MAAN [17], SCRAP and MURK [18] or Mercury [19]. In the specific case of the vector spaces with Euclidean distance, the interval queries can be used to implement the similarity range query.

Several proposed P2P systems are designed to resolve nearest-neighbors queries. The pSearch structure [20] is a P2P information retrieval system for text documents; Distributed Quadtree [21] solves the nearest-neighbors problem in spatial data and SWAM [22] — a family of small-world-based access methods — provides solution for the range and nearest-neighbors search in vector data.

Unfortunately, all the structures mentioned above are designed either for very specific data types or for vector spaces of low dimensionality. Also, the vector space approach uses coordinates to partition the space and, by default, expects the vectors to be compared by Euclidean distance or other L_p metric. Therefore, some measures important for searching in multimedia data types like quadratic-form distance [23] or Earth Mover's Distance [24], require the metric space approach. Moreover, vector-based structures cannot be applied to many important non-vector data types where the similarity is measured by functions such as Hausdorff distance, Jaccard's coefficient, edit distance, etc.

To the best of our knowledge, the four systems analyzed in this paper are the only metric-based distributed data structures published so far. In this respect, this work presents the first and extensive performance comparison of all distributed similarity-search indexes designed for general metric spaces.

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 existing 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. Native Metric Structures: GHT* and VPT*

In this section, we describe two distributed metric index structures — the *GHT** [3] and its extension called the *VPT**. Both of them exploit native metric partitioning principles using them to build a distributed binary tree [25].

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, 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 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 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), 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 information 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 the structures also provide a mechanism called *image adjustment* for updating the imprecise parts of the AST automatically. For more technical details see [3].

3.1.1. Address Search Tree

The AST is a binary search tree based on the Generalized Hyperplane Tree (GHT) [2] in *GHT**, and on the Vantage Point Tree (VPT) [2] 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 Fig. 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 Fig. 1, we select objects o_{10}, o_{12} and promote them to *pivots* of 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 $d(o, o_{10}) \leq d(o, o_{12})$, 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 built accordingly. Fig. 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 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 Fig. 2. In general, this principle also divides 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 than or equal to the specified radius r_1, i.e. if $d(o, o_{11}) \leq r_1$ 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.

3.1.2. Range search

The **Range**(q, r) query search in both the *GHT** and *VPT** structures proceeds as follows. The evaluation starts by

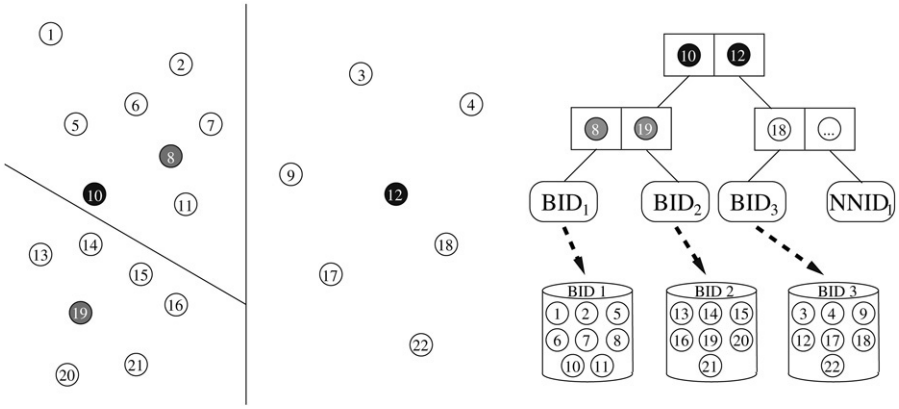

Fig. 1. Address Search Tree with the generalized hyperplane partitioning.

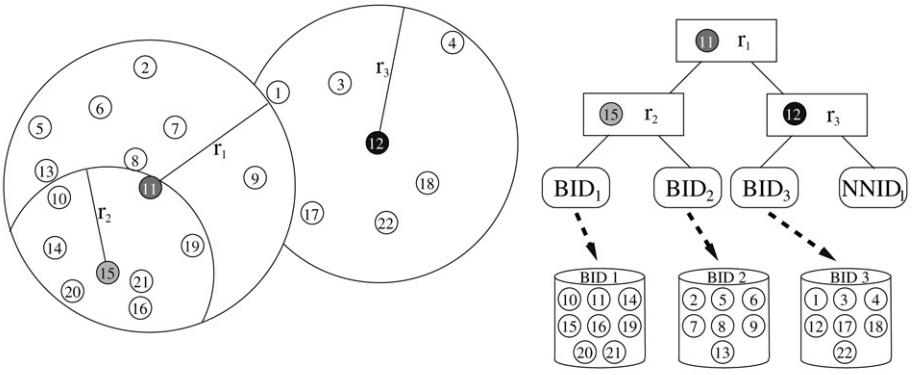

Fig. 2. Address Search Tree with the vantage point partitioning.

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 $\langle p_1, p_2 \rangle$:

$$d(p_1, q) - r \leq d(p_2, q) + r, \tag{2}$$

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

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

$$d(p, q) - r \leq r_p, \tag{4}$$

$$d(p, q) + r > r_p. \tag{5}$$

The right subtree of the inner node is traversed if Condition (2) for the *GHT** or Condition (4) for the *VPT** qualifies. The left subtree is traversed whenever Condition (3) or Condition (5) 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 using according to Definition 2. All qualifying objects together form the query response set.

In order to avoid some distance computations, both the structures apply additional filtering using Eq. (1). For every stored object, the distances to all pivots on the AST path from the root to the leaf with respective bucket are held together with the data object. For example object o_1 in Fig. 1 has associated four numbers — the distances $d(o_1, o_{10}), d(o_1, o_{12}), d(o_1, o_8), d(o_1, o_{19})$ which were evaluated during the insertion of o_1 into bucket BID_1. In the case of *VPT** structure, only half of the distances are stored compared to *GHT**, because only one pivot is present in every 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.

3.2. Metric CAN

In order to manage metric data, the *MCAN* [6] 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* [4] Peer-to-Peer protocol is used to partition the space and for 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 $F : \mathcal{D} \to \mathbb{R}^N$:

$$F(x) = (d(x, p_1), d(x, p_2), \ldots, d(x, p_N)). \tag{6}$$

Coordinates in the virtual vector space designate the placement of object x within the *MCAN* structure. The *CAN*

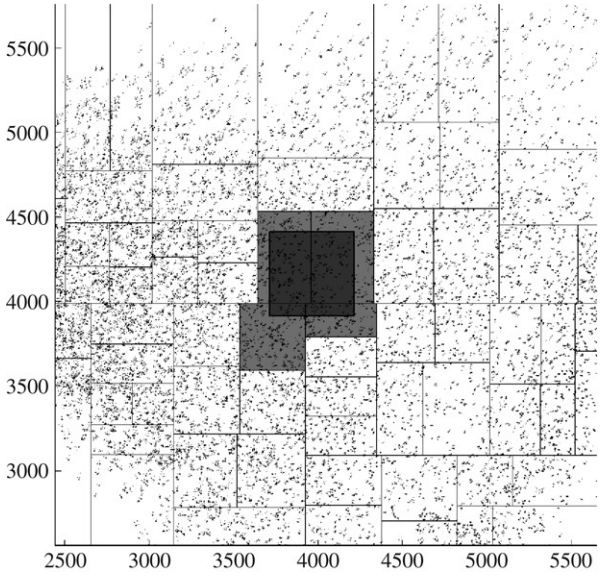

Fig. 3. Example of an MCAN range query.

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. $L^\infty(F(x), F(y)) \leq d(x, y)$, which can be proved using the triangle inequality of the metric function d [6]. Thus, the algorithm for **Range**(q, r) query involves 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 Fig. 3).

In order to further reduce the number of the distances evaluated, *MCAN* uses the additional pivot-based filtering according to Eq. (1). All peers use the same set of pivots: the N pivots from the mapping function F (Eq. (6)) plus additional pivots since N is typically low. All the pivots are selected from a sample dataset using the incremental pivot-selection technique [26].

Routing in *MCAN* works in the same way as in the original *CAN*. Every peer maintains a coordinate-based routing table containing the network identifiers and coordinates of its neighboring peers in the virtual $\mathbb{R}^N$ space. In every step, the routing algorithm passes the query to the neighbor whose region is the closest to the target point in the vector 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 [4].

3.2.1. 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 Eq. (6). The insertion request is then forwarded (using the *CAN* navigation) to the peer that covers value $F(x)$. The receiving peer stores x and splits if its storage capacity limit (or some other defined condition) is reached. The peer's region is split into two parts trying to divide the storage equally. One of the new regions is assigned to a new active peer and the other one replaces the original region.

3.2.2. Range search algorithm

The peer that initiates a **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 $F(q)$. When a peer visited during the routing process intersects the query area, the request is spread to all other relevant peers using a CAN multicast algorithm [27]. A peer intersecting the query region is always reached sooner or later. Every affected peer searches its data storage employing the pivot-filtering mechanism and returns the answer directly to the initiator.

3.3. Metric Chord

Similarly to the *MCAN*, the *M-Chord* [7] 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 [5].

Specifically, this approach exploits the idea of a *vector* index method *iDistance* [28], 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

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

Fig. 4(a) visualizes the mapping schema. Handling a **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 Fig. 4(b).

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 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 [10] (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 $x \in C_i$, $0 \leq i < n$, the *M-Chord* key-assignment formula becomes:

$$m\text{-}chord(x) = h(d(p_i, x) + i \cdot c). \tag{7}$$

3.3.1. 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 [5]. 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

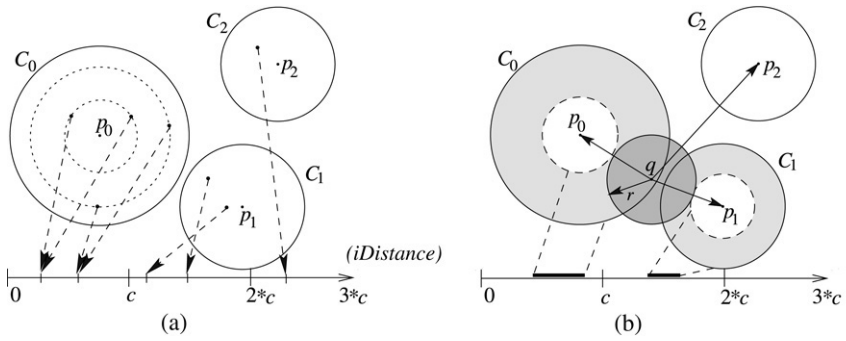

Fig. 4. The principles of iDistance.

Formula (7) and employs *Chord* to forward a store request to the node responsible for key *m-chord*(x) (see Fig. 5(a)).

The nodes store 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 requests a *split*. A new node is placed on the *M-Chord* circle, so that the requester's storage can be split evenly.

3.3.2. Range search algorithm

The node N_q that initiates the **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 is then spread to all nodes covering the particular interval (see Fig. 5(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 (Eq. (1)) improves the pruning of the search space.

3.4. Nearest-neighbors search

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

More precisely, the general **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 kth nearest object found.
(2) Execute the **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 [29] for details.

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

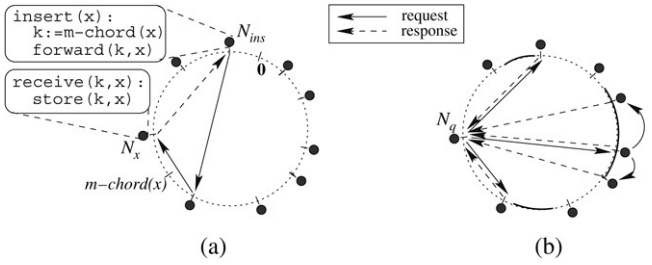

Fig. 5. The insert (a) and range search (b).

4. Evaluation of performance scalability

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 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 size of the query, with respect to the size of the indexed dataset, and considering the number of queries executed concurrently.

4.1. 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 1000 objects and the *MCAN* and *M-Chord* peers had storage capacity of 5000 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:

VEC 45-dimensional *vectors* of extracted color image features. The similarity of the vectors was measured by 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* were of lengths from 3 to 200 characters and are compared by the *edit distance* [30] 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 [31]. This distance function has quite a limited domain of possible values — the returned values are integers between 0 and 100. Processing of DNA sequences is highly computationally intensive and exploiting parallelism is one of possible approaches to this problem [32].

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, bigger datasets consisting of 1,000,000 objects are used (900,000 for **TTL**). As for other settings 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 filtering pivots according to the depth of the AST tree (see Section 3.1).

All presented performance characteristics of query processing have been obtained 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 usually 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 often I/O costs as well) are practically negligible compared to the distance computation time.

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

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 **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% 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 2000 and for the **TTL** and **DNA** datasets from 2 to 20.

In the first group of graphs, shown in Fig. 6(a), we report on the relation between the query radius size and the number of the objects retrieved. 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 datasets. Note that, for example, the biggest radius 2000 in the **VEC** dataset selects almost 10,000 objects (2% of the whole database). 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. 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 Fig. 6(b). 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

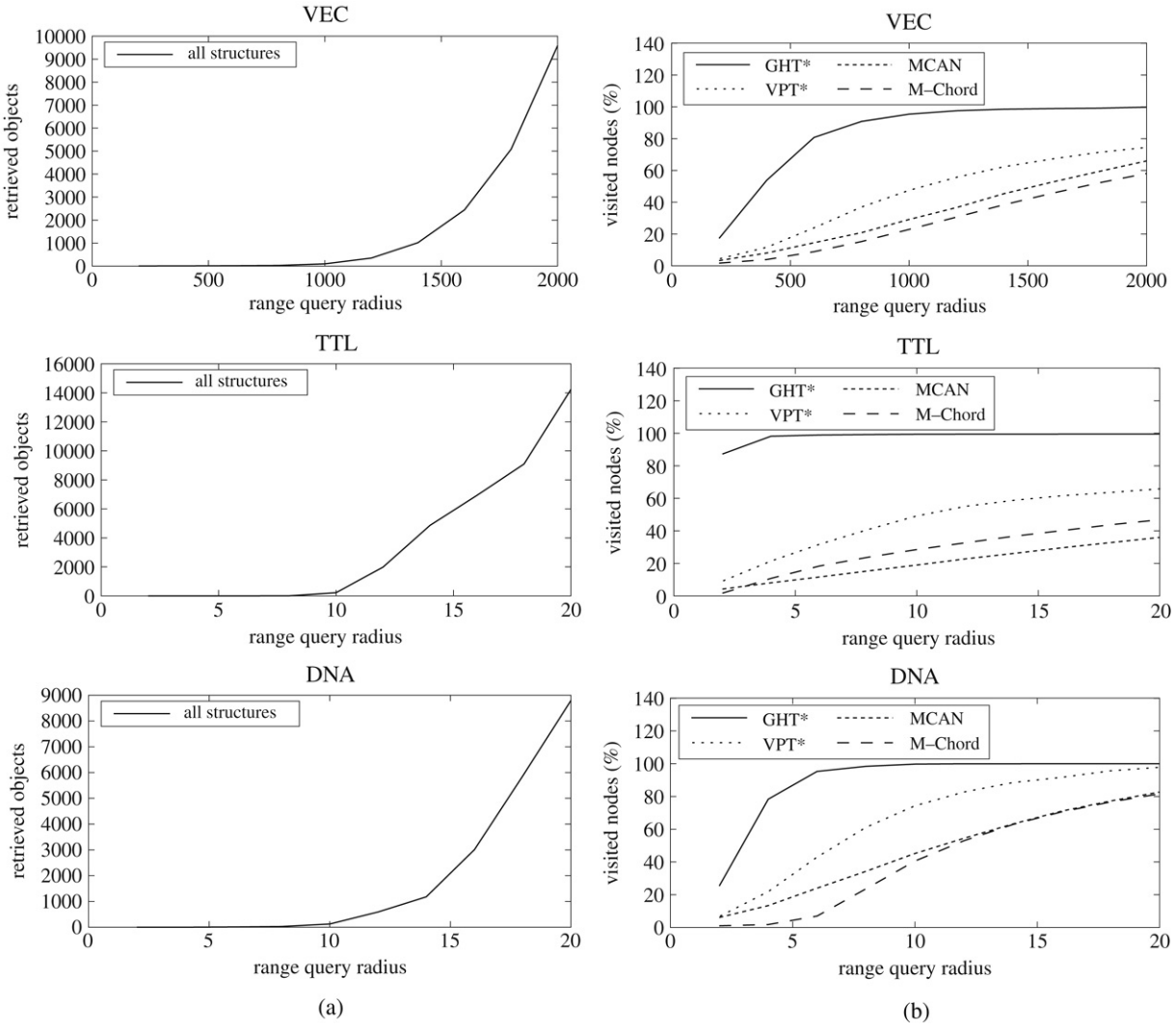

Fig. 6. (a) Number of retrieved objects; (b) Percentage of visited nodes.

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 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 by Fig. 7, shows the computational costs with respect to the query radius. We provide a pair of graphs for each dataset. The graphs on the left (a) report 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 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 to avoid as much distance computations as possible (as mentioned in Section 2). 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.

Looking at the total numbers of distance computations in Fig. 7, we can see that the filtering was rather ineffective in the **DNA** dataset, where the structures have computed the distances

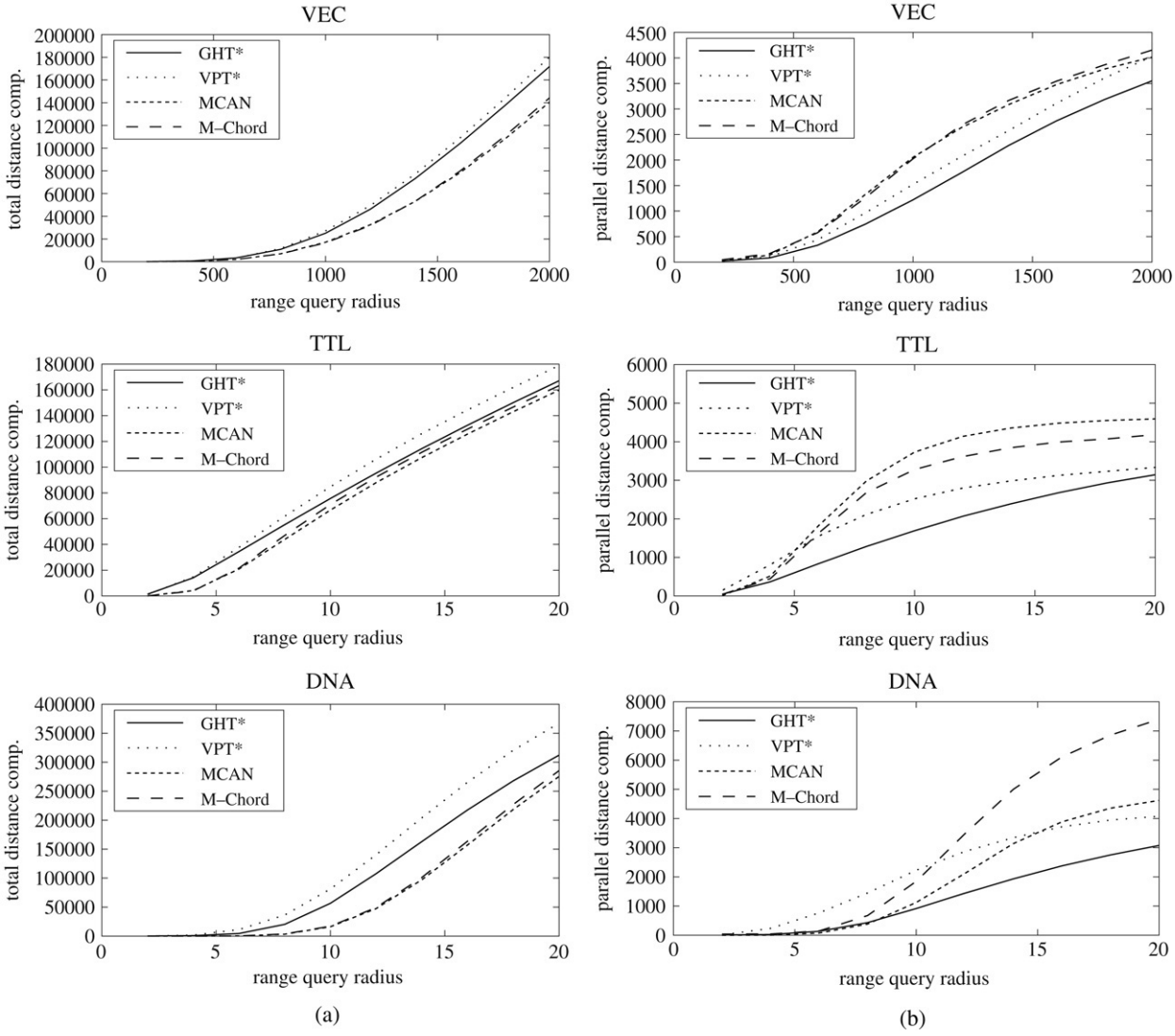

Fig. 7. The total (a) and parallel (b) number of distance computations.

for up to twice as many objects than for the **TTL** and **VEC** datasets. The 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.

Fig. 7(b) illustrates the parallel computational costs of the query processing. We can see that the number 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 Fig. 6(b)), 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 5000 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 5000 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 5000 objects was exceeded.

The last group of measurements in this section, depicted in Fig. 8, reports on the communication costs measured as the total number of messages sent and as the maximal hop count, 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 (Fig. 6(b)) and the total number of messages (Fig. 8(a)) 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.

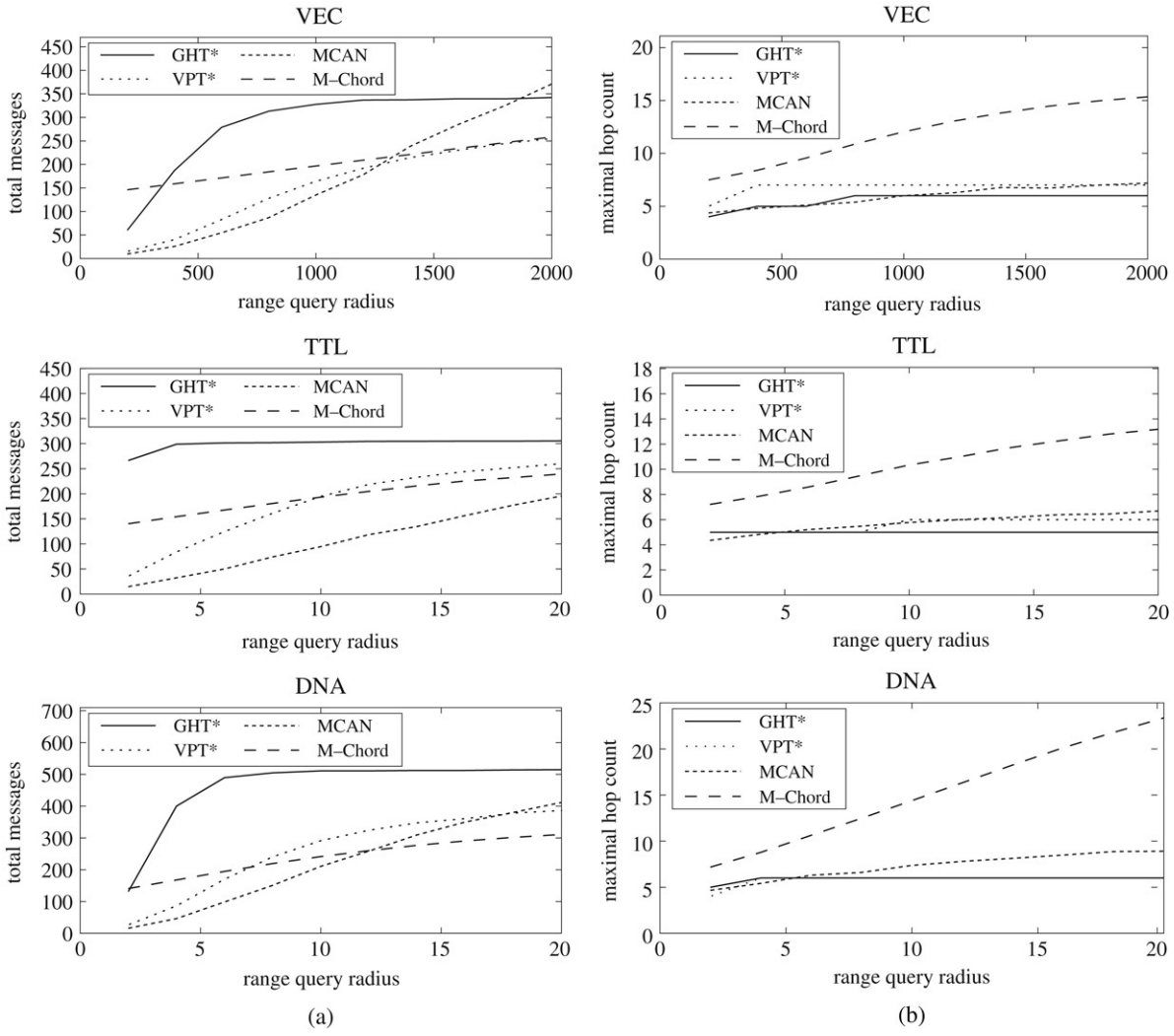

Fig. 8. The total number of messages (a) and the maximal hop count (b).

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. 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.

The parallel costs, i.e. the maximal hop count, are practically constant for different sizes of the radii for all the structures except of *M-Chord* for which it 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 monitored the performance of **Range**(q, r) queries processing on systems storing from 50,000 to 1,000,000 objects. We conducted these experiments for the following radii: 500, 1000 and 1500 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 collected results, therefore, 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 used dataset and the search radius r.

The number of retrieved objects — see graph for radius 10 and the **TTL** dataset in Fig. 9(a) — grows precisely linearly because the data were inserted to the structures in random order.

Fig. 9(b) 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 4.3, the *GHT** partitioning is not balanced, therefore, the query processing is spread over larger number of participating nodes.

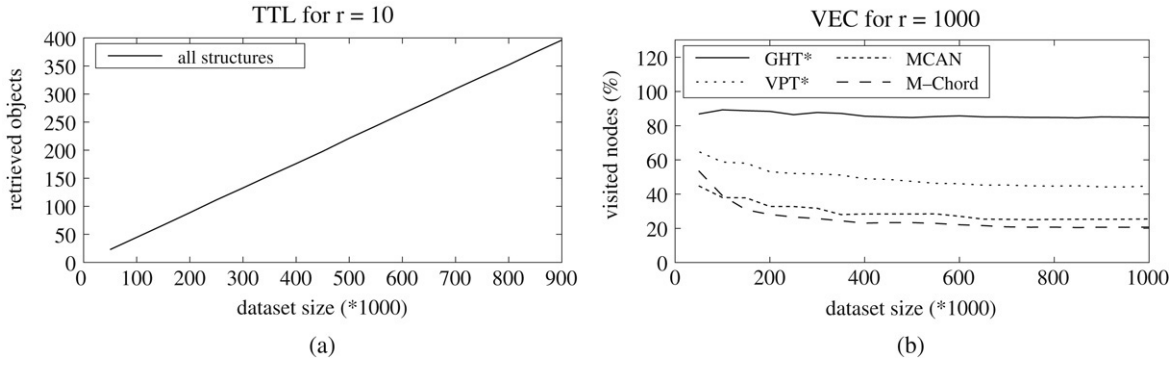

Fig. 9. Retrieved objects (a) and visited nodes (b) for growing dataset.

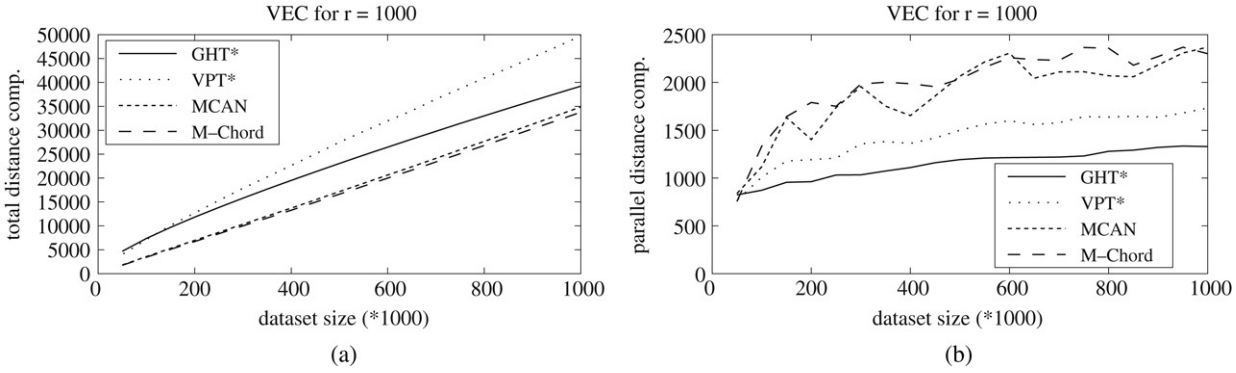

Fig. 10. The total (a) and parallel (b) computational costs for **VEC**.

Fig. 10 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 data volume stored. 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 smaller sets of filtering pivots.

The parallel number of distance computations (Fig. 10(b)) 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 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 to the observable correlation of this graph and Fig. 9(b) — 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 have higher parallel costs then. The noticeable graph fluctuations are caused by quite regular splits of overloaded nodes.

Fig. 11 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 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 Fig. 11(b) may be over the split limit of 5000 objects.

Fig. 12 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.

4.5. 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 [33] of the queries processing.

In the conducted 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 4.4.

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

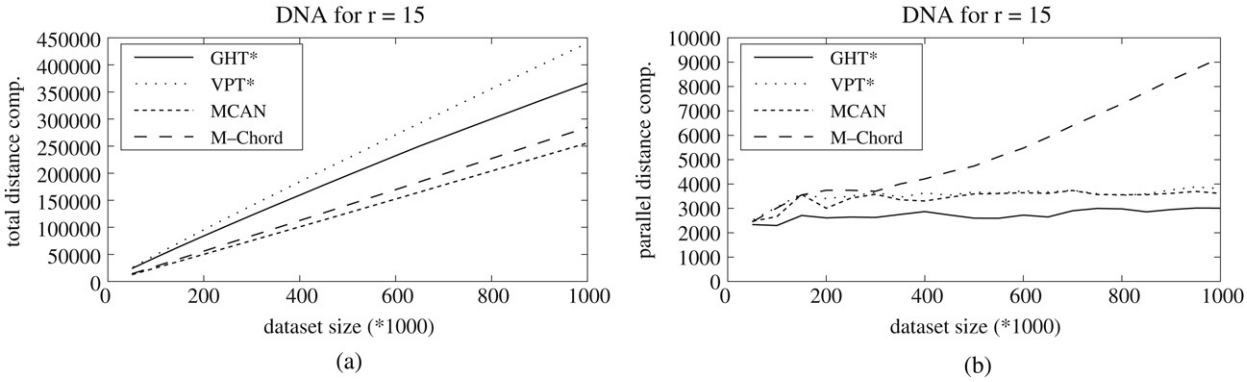

Fig. 11. The total (a) and parallel (b) computational costs for **DNA**.

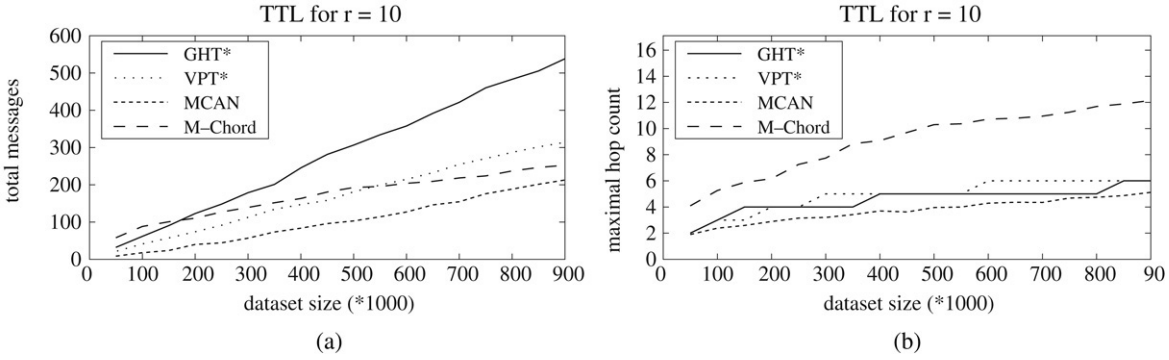

Fig. 12. The total messages (a) and the maximal hop count (b).

value can be also interpreted as the number of queries that can be handled by the systems simultaneously without slowing them down.

Looking at Figs. 13(a), 14(a) and 15(a), 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 similar.

Therefore, differences in the respective interquery improvement ratios, shown in the (b) graphs, are introduced mainly by differences between single-query parallel costs of individual structures. The *M-Chord* and the *MCAN* handle multiple queries better than the *VPT** and notably better than the *GHT**.

The native metric structures employ quite a high number of nodes during the query processing (see Section 4.3). Therefore, multiple queries very likely hit the same nodes, which increases the overall response time. On the other hand, the *M-Chord* and the *MCAN* behave slightly differently. There is a higher probability that different queries incur load at different nodes and, thus, the parallel costs are only marginally increased.

The actual values of the improvement ratio for specific datasets are strongly influenced by the total number of distance computations spread over the nodes (see Fig. 7(a)) and, therefore, the improvement is lower for the **DNA** and **TTL** datasets than for **VEC**.

5. Conclusions

In this paper, we have studied 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) changing 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 special 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 Table 1.

In the table, the *single query* column expresses the power of a corresponding structure to speed up 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.

Table 1
Summary of query processing capabilities

	Single query	*Multiple queries*
*GHT**	Excellent	Poor
*VPT**	Good	Satisfactory
MCAN	Satisfactory	Good
M-Chord	Satisfactory	Very good

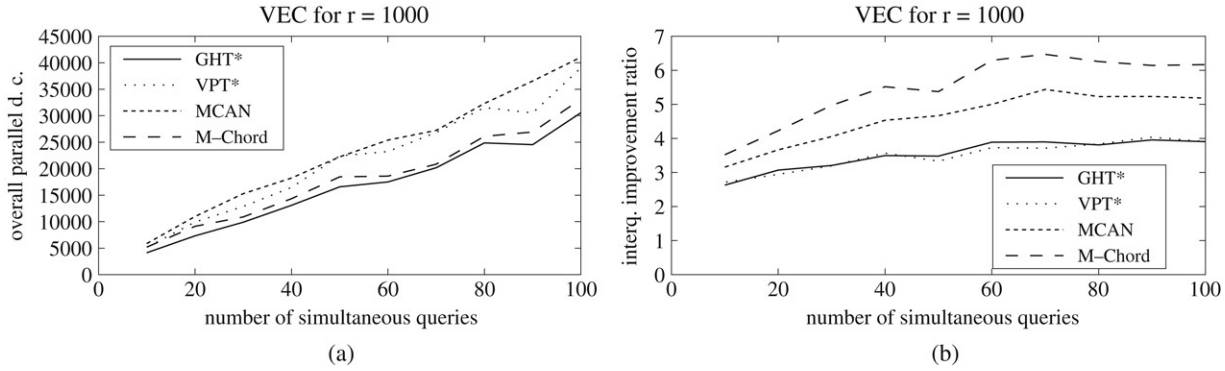

Fig. 13. The overall parallel costs (a) and interquery improvement ratio (b).

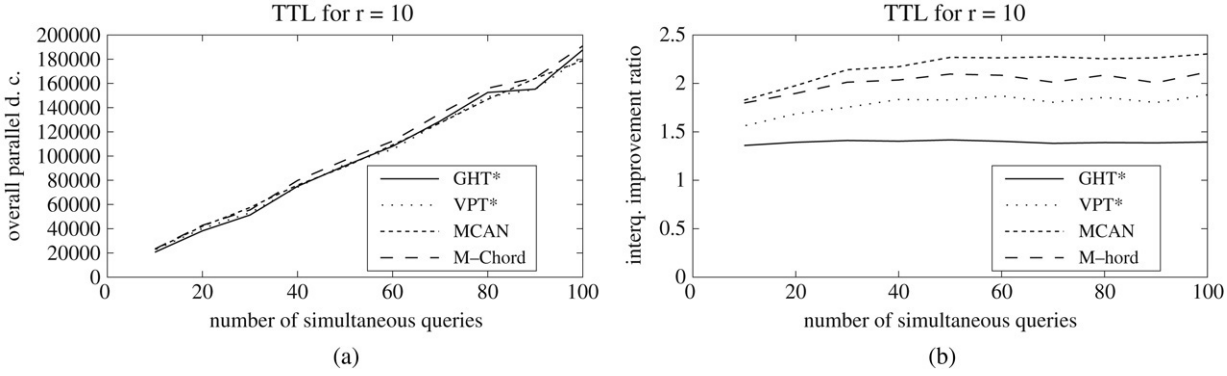

Fig. 14. The overall parallel costs (a) and interquery improvement ratio (b).

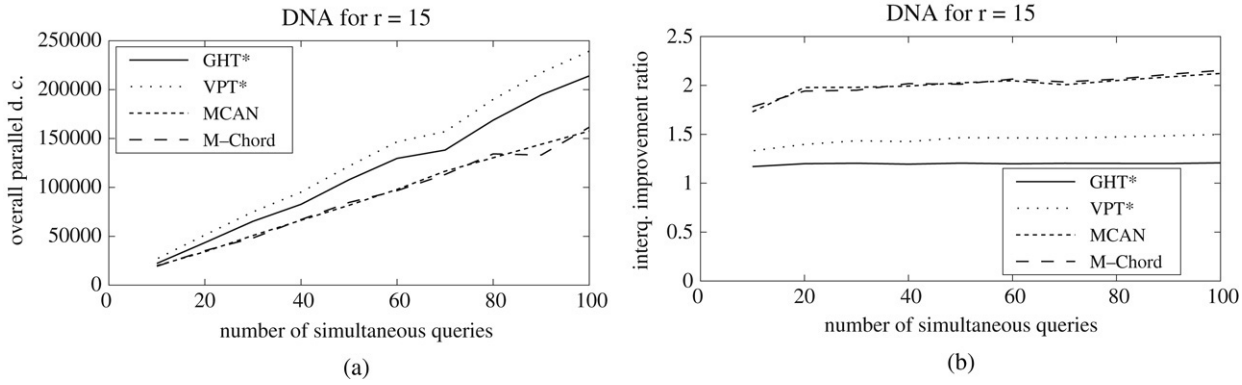

Fig. 15. The overall parallel costs (a) and interquery improvement ratio (b).

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 practically 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 lack of characteristic subset during the 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 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.

Acknowledgements

This research has been partially supported by the national research project No. 1ET100300419 and partially by the Czech Science Foundation grant No. 102/05/H050.

References

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

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

[3] M. Batko, C. Gennaro, P. Zezula, Similarity grid for searching in metric spaces, in: DELOS Workshop: Digital Library Architectures, in: Lecture Notes in Computer Science, vol. 3664/2005, 2005, pp. 25–44.

[4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content addressable network, in: Proceedings of ACM SIGCOMM 2001, ACM Press, 2001, pp. 161–172.

[5] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: A scalable Peer-to-Peer lookup service for internet applications, in: Proceedings of ACM SIGCOMM'01, ACM Press, 2001, pp. 149–160.

[6] F. Falchi, C. Gennaro, P. Zezula, A content-addressable network for similarity search in metric spaces, in: Proceedings of DBISP2P, 2005, pp. 126–137.

[7] D. Novak, 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, ACM Press, 2006.

[8] M. Batko, D. Novak, F. Falchi, P. Zezula, On scalability of the similarity search in the world of peers, in: Proceedings of First International Conference on Scalable Information Systems, INFOSCALE 2006, Hong Kong, May 30–June 1, ACM Press, 2006.

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

[10] G.R. Hjaltason, H. Samet, Index-driven similarity search in metric spaces, ACM Transactions on Database Systems 28 (4) (2003) 517–580.

[11] E. Chávez, G. Navarro, R. Baeza-Yates, J.L. Marroquín, Searching in metric spaces, ACM Computing Surveys 33 (3) (2001) 273–321.

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

[13] K. Aberer, P-Grid: A self-organizing access structure for P2P information systems, Lecture Notes in Computer Science 2172 (2001) 179–194.

[14] J. Aspnes, G. Shah, Skip graphs, in: Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2003, pp. 384–393.

[15] N. Harvey, M.B. Jones, S. Saroiu, M. Theimer, A. Wolman, Skipnet: A scalable overlay network with practical locality properties, in: Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems, USITS '03, Seattle, WA, 2003.

[16] A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke, J. Shanmugasundaram, P-Ring: An index structure for Peer-to-Peer systems, Tech. Rep. TR2004-1946, Cornell University, NY, 2004.

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

[18] P. Ganesan, B. Yang, 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, ACM Press, New York, NY, USA, 2004, pp. 19–24.

[19] A.R. Bharambe, M. Agrawal, S. Seshan, Mercury: Supporting scalable multi-attribute range queries, SIGCOMM Computer Communication Review 34 (4) (2004) 353–366.

[20] C. Tang, Z. Xu, S. Dwarkadas, Peer-to-Peer information retrieval using self-organizing semantic overlay networks, in: Proceedings of SIGCOMM '03, ACM Press, New York, NY, USA, 2003, pp. 175–186.

[21] E. Tanin, A. Harwood, H. Samet, A distributed quadtree index for Peer-to-Peer settings, in: ICDE, IEEE Computer Society, 2005, pp. 254–255.

[22] F. Banaei-Kashani, 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, ACM Press, 2004, pp. 304–313.

[23] T. Seidl, H.-P. Kriegel, Efficient user-adaptable similarity search in large multimedia databases, The VLDB Journal (1997) 506–515.

[24] Y. Rubner, C. Tomasi, L.J. Guibas, A metric for distributions with applications to image databases, in: ICCV '98: Proceedings of the Sixth International Conference on Computer Vision, IEEE Computer Society, Washington, DC, USA, 1998, pp. 59–66.

[25] B. Kröll, 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, ACM Press, Minneapolis, 1994, pp. 265–276.

[26] B. Bustos, G. Navarro, E. Chávez, Pivot selection techniques for proximity searching in metric spaces, Pattern Recognition Letters 24 (14) (2003) 2357–2366.

[27] S. Ratnasamy, M. Handley, R.M. Karp, S. Shenker, Application-level multicast using content-addressable networks, in: NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication, Springer-Verlag, London, UK, 2001, pp. 14–29.

[28] H.V. Jagadish, B.C. Ooi, K.-L. Tan, C. Yu, R. Zhang, iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search, ACM Transactions on Database Systems 30 (2) (2005) 364–397.

[29] M. Batko, C. Gennaro, P. Zezula, A scalable nearest neighbor search in P2P systems, in: Proceedings of DBISP2P, in: Lecture Notes in Computer Science, vol. 3367, 2004, pp. 79–92.

[30] V.I. Levenshtein, Binary codes capable of correcting spurious insertions and deletions of ones, Problems of Information Transmission 1 (1965) 8–17.

[31] S.B. Needleman, 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 (1970) 443–453.

[32] C. Chen, B. Schmidt, An adaptive grid implementation of DNA sequence alignment, Future Generation Computer Systems 21 (7) (2005) 988–1003.

[33] M.T. Özsu, P. Valduriez, Distributed and parallel database systems, ACM Computing Surveys 28 (1) (1996) 125–128.

Michal Batko is a research and development worker at the Faculty of Informatics, Masaryk University, Brno, Czech Republic where he obtained a Ph.D. Degree in computer science. His research activities are concentrated on the efficient searching in large distributed environments with emphasis on the scalability problem. He is focused especially on the problems of distributed similarity search using the metric space approach. As a software developer, he coordinates the development of an extensive similarity searching framework used for creating prototypes of several indexing techniques. Recently, he participated in the EU project called Search on Audio-visual content using Peer-to-Peer Information Retrieval.

David Novak graduated from the Faculty of Informatics, Masaryk University, Brno, Czech Republic with both Bachelor's and Master's Degree in computer science. At the moment, he is a doctoral student at the same faculty and he is working on his thesis on Similarity Search in the Distributed Environment, which is at the center of his professional interests. He participates in two EU projects — DELOS Network of Excellence on Digital Libraries and, recently, project SAPIR, which focuses on searching in multimedia data in the Peer-to-Peer environment.

Fabrizio Falchi is a joint Ph.D. student in Information Engineering at University of Pisa (Italy), and in Informatics at the Faculty of Informatics of Masaryk University of Brno (Czech Republic). In 2003 he received a Research Fellowship from the Networked Multimedia Information System Laboratory of the Information Science and Technologies Institute of the Italian CNR in Pisa, where, from 2006, he is a collaborator. His research interests include similarity search, distributed indexes, multimedia content management systems, content-based image retrieval, Peer-to-Peer systems.

Pavel Zezula is a professor of computer science at the Faculty of Informatics, Masaryk University, Brno, Czech Republic. His professional interests concentrate on storage structures and algorithms for content-based retrieval in non-traditional digital data types and formats, such as the similarity search and the exploration of XML structured data collections. He has a long history of co-operation with the CNR in Pisa, Italy, and has participated in numerous EU projects — the most recent project, SAPIR, explores the search on audio-visual content using Peer-to-Peer information retrieval. He has been a Program Committee Member of several prestigious international conferences (EDBT, VLDB, ACM SIGMOD, and CIKM). His research results appear in major journals such as ACM TOIS, ACM TODS and VLDB Journal, and in the proceedings of leading conferences such as ACM PODS, VLDB, and ACM SIGIR.