Indexing the Distance Using Chord: A Distributed Similarity Search Structure

David Novák and Pavel Zezula

Faculty of Informatics, Masaryk University, Brno, Czech Republic
{xnovak8,zezula}@fi.muni.cz

Abstract. The need of search mechanisms based on data content rather then attributes values has recently lead to formation of the metric-based similarity retrieval. The computational complexity of such retrieval and the large volume of processed data call for distributed processing. In this paper, we propose *chiDistance*, a distributed data structure for similarity search in metric spaces. The structure is based on the idea of a vector-based index method *iDistance* which enables to transform the issue of similarity search into the one-dimensional range search problem. A Peer-to-Peer system based on the *Chord* protocol is created to distribute the storage space and to parallelize the execution of similarity queries. In the experiments conducted on our prototype implementation we study the system performance concentrating on several aspects of parallelism of the range search algorithm.

1 Introduction

In traditional data retrieval, the query specifies a pattern to exactly match attributes of the required data. The present-day systems that manage complex data types (such as images, videos, text documents or DNA sequences) require search mechanisms based on the data content rather then data attributes. Therefore, the field of *content-based* or *similarity* retrieval has made a rapid progress recently. In principle, a similarity query works as follows: given a query data object q, the search process extracts all indexed data objects that are "similar" to q. The similarity of data can be generally defined by a dissimilarity function (*distance function*) d that is measurable for every pair of objects. The data set together with the distance function can be seen as a metric space.

Many metric index structures have been proposed – see recent surveys [1, 2]. In real-life applications, the distance function d is typically expensive to compute. This fact, together with the volume of the data being managed nowadays, lead to the need of distributed processing. Most of the recent effort in the field of distributed indexes for similarity search has concentrated on the vector (attribute) data (see, e.g., [3–5], or SWAM [6]). As far as we know, GHT* index, proposed in [7], is the only published metric-based distributed data structure.

Recently, network architecture paradigms referred to as Peer-to-Peer (P2P) and Grid systems [8] have been gaining in popularity. In short, P2P structures are distributed systems without any hierarchical organization where each node

is running software with equivalent functionality. This concept is attractive for our work because of its scalability and self-organizing nature.

In this paper, we introduce a new distributed data structure called *chiDistance*. It is based on the idea of *Indexing the Distance* (*iDistance*) [9] and generalizes this attribute-based index method to become a metric-based method. Through this concept, the issue of similarity search is transformed into the one-dimensional range search problem. Then, the P2P protocol *Chord* [10] is used to form the distributed structure for *chiDistance*. The *iDistance* search algorithms are generalized to the proposed architecture.

The paper is organized as follows. Section 2 provides background for the metric-based similarity search and describes two techniques that are essential for this paper – *iDistance* and *Chord*. In Section 3, we describe the proposed *chiDistance* data structure in details. Section 4 presents results of the performance experiments and the paper concludes in Section 5 with directions for our future work.

2 Preliminaries

In the scope of our work there are general similarity search structures that are not limited to any specific data set or application. *Metric space* is a suitable structure to model data set and relationships between data objects.

2.1 Metric Space Searching

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

$$\begin{array}{ll} d(x,y) \geq 0 \quad \text{and} \quad d(x,y) = 0 \text{ iff } x = y & (\textit{non-negativity}), \\ d(x,y) = d(y,x) & (\textit{symmetry}), \\ d(x,z) \leq d(x,y) + d(y,z) & (\textit{triangle inequality}). \end{array}$$

Let us define two essential types of similarity queries: the *range query* and the *k nearest neighbors query* [2]. Let $I \subseteq \mathbb{U}$ be a finite set of indexed objects.

Definition 1. *Given an object* $q \in \mathbb{U}$ *and a* maximum search distance r*, the* range query $\mathbf{Range}(q, r)$ *selects all objects* $x \in I$ *such that* $d(q, x) \leq r$*.*

Definition 2. *Given an object* $q \in \mathbb{U}$ *and an integer* $k \geq 1$*, the* k nearest neighbor query $\mathbf{kNN}(q, k)$ *selects the* k *objects* $x \in I$ *which have the shortest distances from* q*.*

The presented data structures focus on these types of similarity queries.

2.2 Indexing the Distance

The *iDistance* [9] is an indexing method for similarity search in vector spaces. It partitions the data space into clusters and selects a reference object (pivot) p_i for each cluster C_i, $0 \leq i \leq k$. Every data object is assigned a one-dimensional *iDistance* key according to the distance to its cluster pivot. Having a cluster separation constant c, the *iDistance* value for an object $x \in C_i$ is

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

If c is large enough then all objects in cluster C_i are mapped to the interval $[i \cdot c, (i+1) \cdot c)$. The data is stored in a B^+-tree according to the *iDistance* values.

Range Query The **Range**(q, r) search algorithm runs separate search procedures for some of the clusters. The cluster C_i is selected for searching if the following condition is satisfied:

$$d(q, p_i) - r \leq \text{max-dist}_i$$

where max-dist$_i$ is the maximal distance between p_i and objects in cluster C_i. Fig. 1 illustrates this condition in two-dimensional space. The cluster searching algorithm examines all objects from C_i which have the *iDistance* value within the interval

$$[d(p_i, q) + i \cdot c - r, d(p_i, q) + i \cdot c + r].$$

The grey areas within the clusters in Fig. 1 represent the space specified by this condition. To examine an object x means to calculate the distance $d(q, x)$ and to add x to the query answer set S if $d(q, x) \leq r$.

***k*-Nearest Neighbor Query** The **kNN**(q, k) algorithm executes a sequence of **Range**(q, r) queries with growing radius r. The condition for adding accessed objects into the answer set differs from the **Range**(q, r) condition. An accessed object x is added into the **kNN**(q, k) answer set S if

$$\text{either} \quad |S| < k \quad \text{or} \quad d(q, x) < d(q, \mathit{farthest}(S, q))$$

where $\mathit{farthest}(S, q)$ is the object from S with the greatest distance from q. The sequence of **Range**(q, r) queries terminates and the **kNN**(q, k) execution is completed when

$$d(\mathit{farthest}(S, q), q) < r \quad \text{and} \quad |S| = k.$$

For each range iteration, the **kNN**(q, k) algorithm stores information about the searched space (taking advantage of the B^+-tree properties) in order not to search any space redundantly.

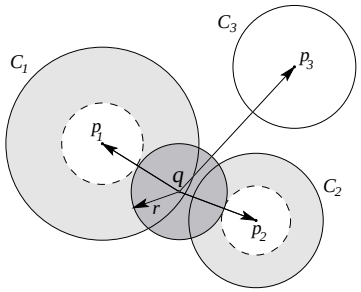

Fig. 1. The *iDistance* search mechanism

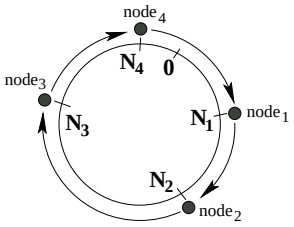

Fig. 2. The *Chord* structure

2.3 Chord

Chord is a purely decentralized structured P2P protocol [10] that provides mechanisms for efficient localization of the node that stores a particular data item (specified by a given search key). It is a message driven dynamic structure that is able to adapt as nodes (cooperating computers) join or leave the system.

The protocol uses *consistent hashing* [11] which uniformly maps the domain of search keys into *Chord* key domain of size 2^m. The parameter m should be large enough to make probability of the hash collisions negligible. Every *Chord* node is assigned a key N_i from the interval $[0, 2^m)$ as well. The node with key N_i is "responsible" for all keys from the interval $(N_{i-1}, N_i] \pmod{2^m}$ – see Fig. 2 for visualization. A node stores objects with the keys the node is responsible for.

Every node maintains a routing table called *finger table* which stores physical addresses of up to m other nodes [10]. The node knows physical addresses of its predecessor and successor as well.

Due to the uniformity of the *Chord* key domain distribution, *Chord* structure has the following properties:

- in an n-node system, the node responsible for a given key is located via $\mathcal{O}(\log n)$ number of messages to other nodes (number of hops);
- the load of particular nodes (number of objects stored in the node) is approximately equal for all nodes.

These properties are very important for performance of the data structure proposed in this paper and are referenced below.

3 ChiDistance

In this section, we introduce *chiDistance* – a new distributed data structure and algorithms for similarity search in metric spaces. The system is based on the idea of *iDistance* (see Section 2.2) and extends it as follows:

- *chiDistance* employs metric pivot selecting techniques to generalize the *iDistance* applicability to metric spaces. Having a finite sample data set $S \subseteq \mathbb{U}$,

a set of k pivots is selected and then *Voronoi*-like partitioning [2] is used to identify the clusters (see Section 3.1 for details).
- It uses the *Chord* protocol and the *chiDistance* key assignment mechanism to distribute the storage space among arbitrary number of cooperating nodes (Section 3.3).
- It provides distributed search algorithms for the Range and kNN queries that parallelize the time-consuming queries execution (Section 3.4).

As mentioned in Section 2.3, the *Chord* protocol assumes the uniform distribution of the key domain to keep its efficiency. Because *iDistance* distribution is strongly non-uniform, the *iDistance* values must be transformed by a hash function with a uniform distribution. Then the *Chord* protocol can be used to divide the storage space among the cooperating nodes. Section 3.2 describes the mechanism of finding the uniform key transformation.

Both, the pivot selection and the phase of determining the uniform hash function, must proceed during the initialization phase of the algorithm. When pivots $\{p_1, \ldots, p_k\}$ are selected, data set is divided into clusters $\{C_1, \ldots, C_k\}$, and the uniform transformation h is found, the *chiDistance* value of an object $x \in C_i$ is computed as follows:

$$chi(x) = h(d(p_i, x) + i \cdot c). \tag{1}$$

Fig. 3 illustrates the process of the algorithm initialization and *chiDistance* computation.

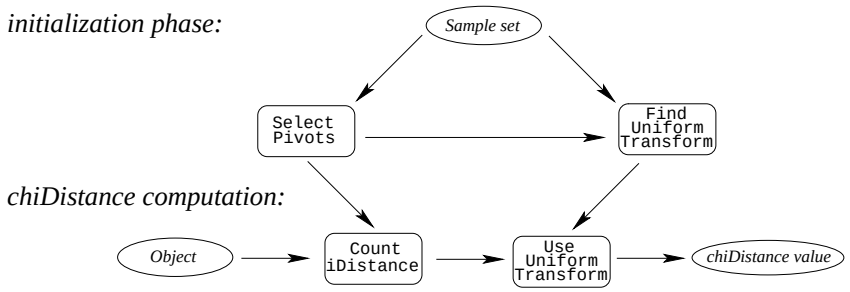

Fig. 3. Algorithm initialization and *chiDistance* computation process

3.1 Pivots Choosing Procedure

As mentioned in Section 2.2, the *iDistance* algorithm uses vector space properties to partition the data space and then to select the reference points (pivots) of identified partitions. In order to generalize the applicability of the method to metric spaces, we use a metric-based technique to efficiently select a set of pivots.

Then, having a set of k pivots $\{p_1, \ldots, p_k\}$, we divide objects from I (the set of indexed objects) into clusters $C_1, \ldots, C_k$ as follows:

$$C_i = \{x \in I | d(p_i, x) < d(p_j, x), 1 \leq j \leq k, j \neq i\}.$$

This partitioning technique is referred to as Voronoi-like partitioning [2].

Generally, the similarity search algorithms eliminate some data objects from the search process without computing their distances to the query object. For pivot-based data structures, the main objective of finding a suitable set of pivots is to increase the effectiveness of such pruning of the search space.

Let us summarize the **Range**(q, r) search algorithm described in Section 2.2. An object x from cluster C_i (with pivot p_i) is accessed if its *iDistance* value ($iDist(x) = d(p_i, x) + i \cdot c$) is within the interval

$$[d(p_i, q) + i \cdot c - r, d(p_i, q) + i \cdot c + r].$$

This condition can be reformulated as follows: An object $x \in C_i$ can be eliminated without accessing if

$$|d(p_i, x) - d(p_i, q)| > r.$$

Thus, the higher the value $|d(p_i, x) - d(p_i, q)|$ for objects $x \in C_i$, $\forall i \in \{1, \ldots, k\}$ the more effective the search algorithm.

The *chiDistance* pivot selection technique is based on the general technique described in [12] which is tuned to fit our search algorithm. Having a sample set of data objects $S \subseteq \mathbb{U}$, we try to choose a set of k pivots $\{p_1, \ldots, p_k\}$ from S in order to "maximize" the function $|d(p_x, x) - d(p_x, y)|$ for every $x, y \in S$ where $p_x \in \{p_1, \ldots, p_k\}$ is the pivot closest to object x. One way to do this is to maximize the mean of distribution of $|d(p_x, x) - d(p_x, y)|$ on S. Let us denote this mean value as $\mu_{\{p_1,\ldots,p_k\}}$ for set of pivots $\{p_1, \ldots, p_k\}$. Now, we can say that $\{p_1, \ldots, p_k\}$ is a better set of pivots than $\{p'_1, \ldots, p'_k\}$ when

$$\mu_{\{p_1,\ldots,p_k\}} > \mu_{\{p'_1,\ldots,p'_k\}}.$$

The $\mu_{\{p_1,\ldots,p_k\}}$ value is estimated in the following way: a set of n pairs $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ is chosen randomly from the sample set S. For each pair (x_i, y_i) the closest pivot p_{x_i} is found and value $v_i = |d(p_{x_i}, x_i) - d(p_{x_i}, y_i)|$ is computed. The value $\mu_{\{p_1,\ldots,p_k\}}$ is estimated as

$$\mu_{\{p_1,\ldots,p_k\}} = \frac{1}{n} \sum_{i=1}^{n} v_i.$$

We need $k + 1$ distance computations to obtain value v_i for each pair of objects. Therefore, $n \cdot (k + 1)$ distance computations are needed to estimate $\mu_{\{p_1,\ldots,p_k\}}$.

Now, knowing how to compare two sets of pivots, we can define the selection technique itself. We use the incremental algorithm described in [12]. First, pivot p_1 is chosen from a set of s candidates (selected from the sample set S) such

that p_1 has the maximum $\mu_{\{p_1\}}$ value. Then, a second pivot p_2 is selected from another set of s objects such that $\mu_{\{p_1,p_2\}}$ is maximal considering p_1 fixed. This process is repeated k-times to get a set of k pivots.

In every iteration of the described algorithm, we can store the closest pivot p_{x_i} to x_i and their distance $d(p_{x_i}, x_i)$ (for every $1 \leq i \leq n$). Thus, only $d(p_j, x)$ and (maybe) $d(p_j, y)$ values need to be computed when the j^{th} pivot is added. Therefore, maximally $2ns$ distance computations are needed while adding a new pivot. Repeating this step k times, the total cost of the pivot selection process is $2kns$ distance computations.

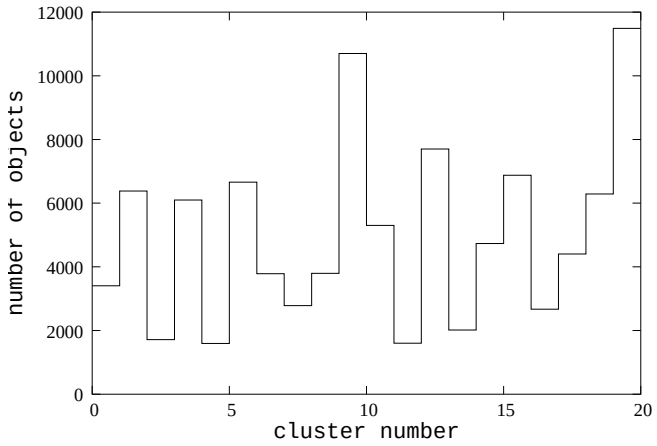

Fig. 4. Number of data objects in clusters

Fig. 4 shows an example of how the data is partitioned to clusters identified by the described method (number of objects in each cluster). The data set is composed of 100,000 three-dimensional vectors (see Section 4 for details about the data set). The sample set consists of 1,000 object pairs ($n = 1000$); number of pivot candidates in every algorithm step is 100 ($s = 100$) and the number of selected pivots is twenty ($k = 20$).

3.2 Uniform Order-Preserving Transformation

The *Chord* routing protocol assumes a uniform layout of the nodes (peers) on the key space circle to keep the property of logarithmic navigation through the structure (Section 2.3). The objective our structure would like to reach is the balanced storage load of the nodes (volume of data stored in the nodes). In order to meet these two criteria, the key domain should have uniform distribution.

The distribution of the *iDistance* domain is strongly non-uniform. Fig. 5 visualizes a typical *iDistance* distribution using 20 pivots. This domain can be

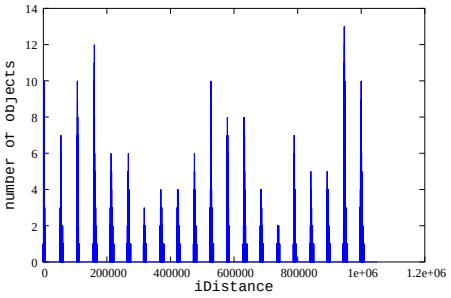

Fig. 5. Distribution of the *iDistance* domain

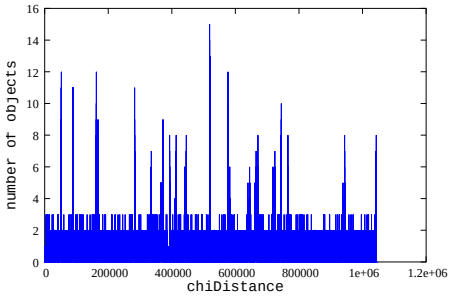

Fig. 6. Distribution of the *chiDistance* domain – h function used

transformed by a hash function with a uniform distribution. Of course, such transformation must be order-preserving to keep *iDistance* properties.

We use a technique described in [13] designed for finding order-preserving uniform transformations between arbitrary domains. Let us denote the desired transformation $h : [0, A] \longrightarrow [0, B]$ (in our case the $[0, A]$ interval is the *iDistance* domain and the $[0, B]$ interval is the *chiDistance* domain of optional size).

First, this method divides target space $[0, B]$ into p intervals of the same length where p is parameter of the algorithm (its precision). This partitioning gives a sequence of values $b_0, b_1, \ldots, b_p$ from $[0, B]$. Then, having an non-decreasing sample sequence $a_1, a_2, \ldots, a_n$ of keys from $[0, A]$, we select the following $p + 1$ values from this sequence:

$$a_0, a_{\lceil \frac{n}{p} \rceil}, \ldots, a_{\lceil i \cdot \frac{n}{p} \rceil}, \ldots, a_n.$$

Let us denote $a_{(i)}$ element $a_{\lceil i \cdot \frac{n}{p} \rceil}$ for every $0 \leq i \leq p$. The desired transformation h maps $h(a_{(i)}) = b_i$ for every $0 \leq i \leq p$. These values are fixed and the h values for all other keys from $[0, A]$ are computed as the linear interpolation of them:

$$h(x) = (x - a_{(i-1)}) \cdot \frac{b_i - b_{i-1}}{a_{(i)} - a_{(i-1)}}, \text{ for } x \in (a_{(i-1)}, a_{(i)}).$$

Thus, h is the piecewise-linear transformation. The "quality" of the uniformity of the image domain distribution depends on the precision factor p. Fig. 6 shows the distribution of the domain from Fig. 5 after applying function h. The precision factor $p = 200$; the sample set $\{a_0, a_1, \ldots, a_n\}$ of size $n = 5000$ has been selected randomly from the whole data set of size 100,000. Size of both domains is 2^{20}.

3.3 *chiDistance* Data Structure

The proposed data structure is a message driven structured P2P system based on the *Chord* routing mechanism. Each node of the structure is logically composed of two layers – *Chord* layer and *chiDistance* layer. The topology of the network

is given by the *Chord* protocol (the *Chord* layers of the nodes communicate with each other to form and maintain the structure).

When a new node wants to *join* the structure, it contacts an already participating node. It is assigned a key from the *chiDistance* domain and it receives data objects with keys from the interval of its responsibility (Section 2.3). The keys for nodes are chosen uniformly at random. The joining node receives the "*chiDistance* configuration" as well – the selected pivots and the uniform transformation h. To *leave* the system, a node notifies its successor node and sends all stored data to it.

The *chiDistance* layer forms the interface of the system on every node. When receiving an *insert*/*delete*/*exact-match* operation request, first, it calculates the *chiDistance* value for the object passed by the operation (Equation 1). Then the *Chord* layer locates the node that is responsible for the computed key and the operation request is passed to that node to store/delete/get the object.

Due to usage of the constant c in the *chiDistance* key formula (1), the domain consists of separated segments that correspond to particular clusters. It may happen that one node is responsible for intervals of keys belonging to several clusters. Vice versa, the interval corresponding to a cluster can be divided among several adjacent *chiDistance* nodes.

Every node stores the data separately for every covered cluster. The data objects are stored in a Red-Black tree based structure that provides guaranteed $\log(n)$ time cost for *get*, *put* and *remove* operations and provides $range(from, to)$ operation as well. This storage policy is convinient for the search algorithm.

3.4 Range Search Algorithms

The *chiDistance* $\mathbf{Range}(q, r)$ search algorithm follows the basics of *iDistance* algorithm and distributes it parallelizing the query execution. The query processing starts on the initiating node and then spreads over the other nodes.

The algorithm searches the clusters $C_1, C_2, \ldots, C_k$ separately. The segment of data to be searched within each cluster C_i is dependent on the *chiDistance* key that would be assigned to object q if object q was in cluster C_i:

$$chi_i(q) = h(d(p_i, q) + i \cdot c).$$

This value is computed for every cluster C_i and the requests for clusters search are sent to the nodes that are responsible for these chi_i keys – let us denote these nodes N_i.

Within cluster C_i, all objects with *chiDistance* keys from the following interval must be examined:

$$[h(d(p_i, q) + i \cdot c - r), h(d(p_i, q) + i \cdot c + r)].$$

Thus, the N_i node explores all data objects from this interval that are stored in N_i. But objects from cluster C_i can be stored in several adjacent nodes so – if N_i is not responsible for the terminal points of this interval – the search request is forwarded to the predecessor and/or successor node(s) of N_i. In order

to parallelize the execution, node N_i first forwards the requests and then searches its own data space.

For every accessed data object x, the distance $d(q, x)$ is computed and if $d(q, x) \leq r$ then x is added to the **Range**(q, r) query answer set S. The partial answer sets from all visited nodes are returned to the query originating node and are joined together to form the final anwer set of the range query.

4 Performance Evaluation

In this section, we present results of experiments we conducted on our prototype implementation of *chiDistance* data structure. The system forms a logical overlay network independent of the physical location of the participating nodes. The communication among the nodes is realized via messages using the UDP and TCP communication protocols.

4.1 The Data Sets and Parameter Settings

We executed our experiments on two data sets. The first is a set of artificially generated three-dimensional vectors of real numbers with the L_2 (Euclidian) metric distance function (VEC). This data set has a uniform distribution of distances between objects and all objects have the same size.

The second data set is a real-life set of sentences from the Czech national corpus with the *edit distance* function as the metric (TXT). The length of the sentences varies significantly and the distribution of the distances is rather skewed – most of the distances are within a small range of values.

Both data sets consist of 100,000 objects. All experiments are performed on a structure formed by up to 100 cooperating nodes running on PCs connected by a high-speed local network. The first executed process is given a sample data set of 5,000 objects randomly selected from the whole data set. This sample set is used during the pivot selection (Section 3.1) and then to determine the uniform hash function h (Section 3.2). The number of pivots/clusters is fixed to $k = 20$ – exploring the influence of this parameter to the performance is part of our future work. The precision parameter p of the hash function h is set to 200 which seems to be sufficient for the size of the used *chiDistance* domain $[0, 2^{20})$.

4.2 Domain Coverage and Load Factor

Every node joining the system is assigned a key from the *chiDistance* domain and covers a domain segment $(predecessor, node]$ (Section 3.3). Fig. 7 shows the size of intervals covered by each of 100 nodes (the domain size is $2^{20} = 1048576$).

One of the characteristics influencing the system performance is how the data set is distributed among the nodes. Fig. 8 shows number of data objects stored in each of the nodes (load factor). As discussed in Section 3.2, the distribution of the *chiDistance* domain is broadly uniform and therefore the load factor rather copies the domain coverage from Fig. 7.

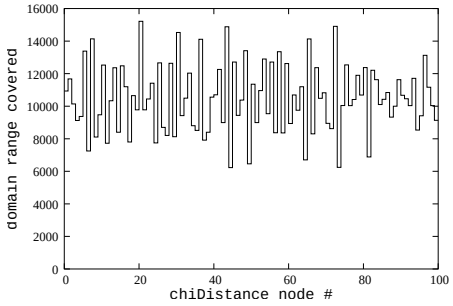

Fig. 7. Segments of *chiDistance* domain covered by individual nodes

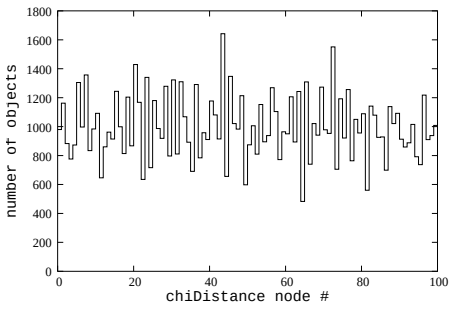

Fig. 8. Number of data objects stored by individual nodes

4.3 Key Locating – Insert, Delete, Exact-match

The *Chord* routing mechanism used by our system is able to locate the node responsible for given *chiDistance* key in $\mathcal{O}(\log n)$ number of hops (where n is the number of nodes in the system). The key locating mechanism is used by most of the operations on the structure and its complexity is very important. Fig. 9 shows the experimental results – the hop count with respect to the number of nodes in the system (n). The values were obtained as an average over 100 operations.

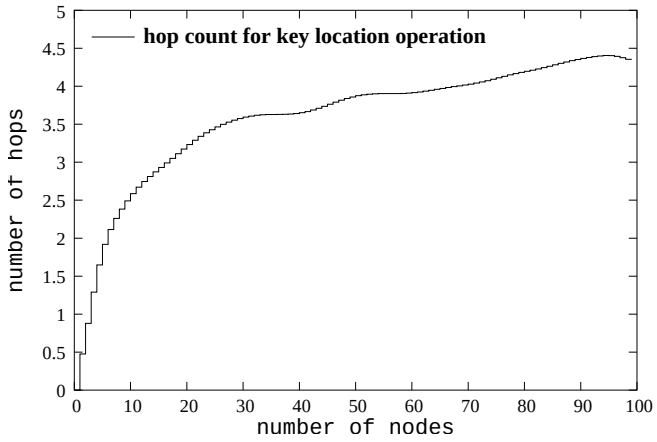

Fig. 9. Number of hops to locate the node responsible for given *chiDistance* key.

The complexity of the key localization is the complexity of the *insert*, *delete*, and *exact-match* operations as well.

4.4 Range Search Performance

In this section, we present results of experiments analyzing the performance of the **Range**(q, r) search algorithm. To measure the computational cost of the operation, we use the number of distance computations on the participating nodes. This performance metric is common for similarity search algorithms because the actual cost of the distance computation may differ signifiantly for various distance functions. In the experiments, we have neglected the inter-nodes communication time because the distance computations are more time consuming than sending a message between nodes.

Thus, most of the analyses in this section are based on observing how the distance computations are distributed on the set of nodes (under various circumstances). The first observation is that the total number of distance computations (*total cost*) remains the same for any number of nodes as well as for the centralized *iDistance* method. This quantity predicates generally about this search method efficiency with respect to our pivot choosing technique.

One of the very important contributions of distributed structures is the parallelism of a query execution. In our experiments, we observe several aspects of the *intraquery* and *interquery* parallelism [14]. We quantify the intraquery parallelism as the parallel computational cost of one query execution – maximal number of distance computations performed in sequence (the *parallel cost*).

Fig. 10 presents results of the experiment which measured the parallel cost with respect to the growing search radius. The second curve in the graph shows the total number of distance computations (divided by 10 to see both shapes properly). All values presented in this section are taken as an average over 100 queries with different query objects randomly chosen from the data set.

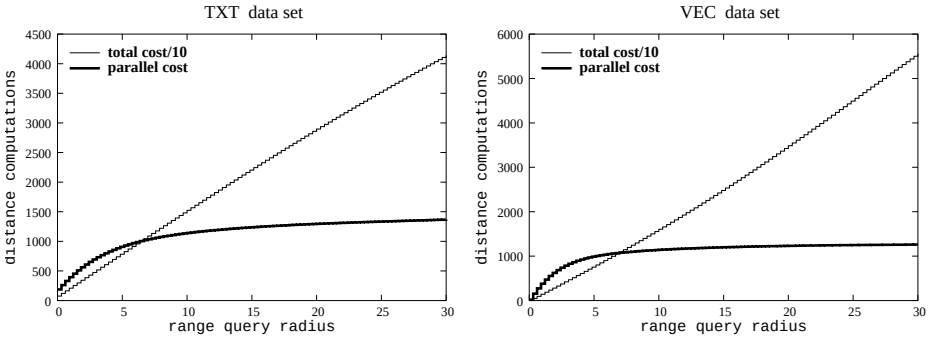

Fig. 10. The total and parallel cost for range queries increasing the radius.

This experiment results show that, for larger query radii, the parallel cost corresponds to the nodes load factor – some nodes access all the data objects they store. Let us denote C_q the cluster which the query object belongs to. The

search algorithm principles imply that (for larger radii) most of the objects in cluster C_q must be accessed. Very likely, some node(s) store(s) only objects from cluster C_q because, in our setting, the number of nodes is higher than the number of pivots/clusters. This causes the rapid increase of the parallel cost. Obviously, most of the nodes are computationally loaded significantly less than the parallel cost is, because the average cost per server (total cost/100) is lower. This fact is analyzed by the interquery parallelism experiments.

The interquery parallelism refers to the ability of the system to accept multiple queries at the same time. Let us denote the set of active nodes $\{n_1, \ldots, n_m\}$ and the numbers of distance computations on node n_i as $cost_i^1, cost_i^2, \ldots, cost_i^k$ for the sequence of k queries. We measure the interquery parallelism through summation of the number of distance computations over a sequence of range queries (on each node separately):

$$\textit{inter-cost}_i = \sum_{j=0}^{k} cost_i^j.$$

Maximizing this value over the set of nodes $\{n_1, \ldots, n_m\}$ we get a total parallel cost of the sequence of k queries:

$$\textit{inter-cost} = \max\{\textit{inter-cost}_1, \ldots, \textit{inter-cost}_m\}.$$

Fig. 11 shows the *inter-cost* value for growing number of parallel queries (k) and for selected radii. The query objects were chosen at random and the graph values are taken as an average over 10 executions of different sets of k queries. As discussed above, mainly the nodes storing objects from cluster C_q are significantly computationally loaded during the $\mathbf{Range}(q, r)$ execution. Thus, the *inter-cost* is significantly smaller than k-multiple of the parallel cost of one range query (with corresponding radius).

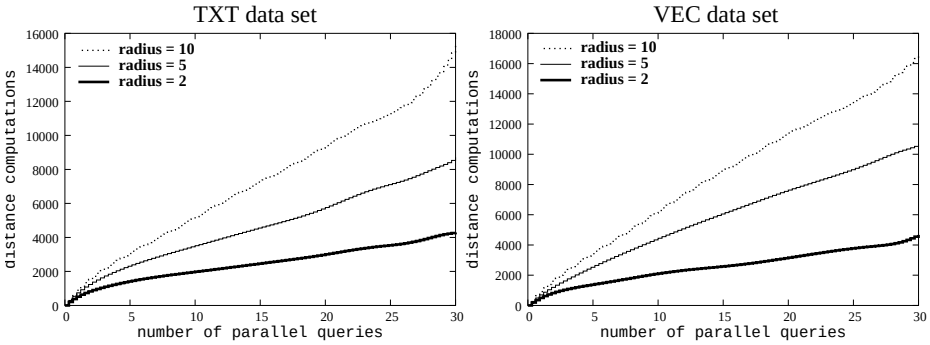

Fig. 11. The interquery parallelism for growing number of parallel queries

5 Conclusions and Future Work

So far, the field of distributed index structures for metric data has not been much investigated. We consider this topic very relevant for needs of the preset-day applications processing large volumes of digital data. We propose *chiDistance*, a distributed data structure for similarity search in metric spaces. The structure is based on the idea of *iDistance* index method. The applicability of *iDistance* is generalized from vector spaces to metric spaces by a pivot selection technique tuned in order to fit the *chiDistance* search algorithm. A Peer-to-Peer structure based on the *Chord* protocol is created to distribute the storage space and to parallelize the execution of similarity queries.

We present results of performance experiments conducted on our prototype implementation and on two data sets. Among other things, we have studied several aspects of parallelism of the range search algorithm. The results show that the computational cost on participating nodes varies but the cost on a single node is always upper bounded by the number of data objects stored by the node. Recall that the data is quite uniformly distributed among the nodes. At the same time, very good interquery parallelism is achieved when processing a set of range queries in parallel.

Our future work will concern the $\mathbf{kNN}(q, k)$ search algorithm. As well, we want to study the influence of number of pivots to the search algorithms performance. In this paper, we have considered the presented structure rather static by using fixed quantity of sources (nodes) regardless of the volume of stored data. In future, we want to study the scalability of the system by adding nodes as the volume of stored data increases. Finally, we plan to conduct tests that would compare the performance of GHT* [7] and *chiDistance* – both implemented over the same network infrastructure. We see this as a unique opportunity to compare structures and algorithms under the very same conditions.

References

1. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces. ACM Trans. Database Syst. **28** (2003) 517–580
2. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. **33** (2001) 273–321
3. Koudas, N., Faloutsos, C., Kamel, I.: Declustering spatial databases on a multi-computer architecture. In: EDBT. (1996) 592–614
4. Gupta, A., Agrawal, D., El Abbadi, A.: Approximate range selection queries in peer-to-peer systems. In: Proceedings of the First Biennial Conference on Innovative Data Systems Research, Asilomar, California, United States (2003)
5. Tanin, E., Harwood, A., Samet, H.: A distributed quadtree index for peer-to-peer settings. In: ICDE. (2005)
6. Banaei-Kashani, F., Shahabi, C.: 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) 304–313

7. Batko, M., Gennaro, C., Zezula, P.: Scalable similarity search in metric spaces. In: Proceedings of the DELOS Workshop on Digital Library Architectures: Peer-to-Peer, Grid, and Service-Orientation, Edizioni Libreria Progetto, Padova (2004) 213–224
8. Androutsellis-Theotokis, S., Spinellis, D.: A survey of peer-to-peer file sharing technologies (2002)
9. Yu, C., Ooi, B.C., Tan, K.L., Jagadish, H.V.: Indexing the distance: An efficient method to KNN processing. In: VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy, Morgan Kaufmann (2001) 421–430
10. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of ACM SIGCOMM, ACM Press (2001) 149–160
11. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D.: Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In: STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, ACM Press (1997) 654–663
12. Bustos, B., Navarro, G., Chávez, E.: Pivot selection techniques for proximity searching in metric spaces. In: SCCC 2001, Proceedings of the XXI Conference of the ChileanComputer Science Society, IEEE CS Press (2001) 33–40
13. Garg, A.K., Gotlieb, C.C.: Order-preserving key transformations. ACM Trans. Database Syst. **11** (1986) 213–234
14. Özsu, M.T., Valduriez, P.: Distributed and parallel database systems. ACM Comput. Surv. **28** (1996) 125–128