A Content–Addressable Network for Similarity Search in Metric Spaces[⋆]

Fabrizio Falchi[1], Claudio Gennaro[1], and Pavel Zezula[2]

[1] ISTI-CNR
Pisa, Italy
fabrizio.falchi,gennaro@isti.cnr.it
[2] Masaryk University
Brno, Czech Republic
zezula@fi.muni.cz

Abstract. In this paper we present a scalable and distributed access structure for similarity search in metric spaces. The approach is based on the Content–addressable Network (CAN) paradigm, which provides a Distributed Hash Table (DHT) abstraction over a Cartesian space. We have extended the CAN structure to support storage and retrieval of more generic metric space objects. We use pivots for projecting objects of the metric space in an N-dimensional vector space, and exploit the CAN organization for distributing the objects among computer nodes of the structure. We obtain a Peer–to–Peer network, called the MCAN, which is able to search metric space objects by means of the similarity range queries. Experiments conducted on our prototype system confirm full scalability of the approach.

1 Introduction

The proliferation of digital contents such as video, images, or text imposes the use of access methods for efficiently storing and retrieving this information. The concept of similarity searching based on relative distances between a query and database objects has become a solution for a number of application areas, e.g. data mining, signal processing, geographic databases, information retrieval, or computational biology. This approach formalizes the problem by the mathematical notion of the *metric space* [6], so the data elements are assumed to be objects from a metric space domain where only pairwise distances between the objects can be determined by respective distance function.

However, the need to deal with larger and larger, possibly distributed, archives requires an access structure to speedup the retrieval. Unfortunately, the use of single–site access structures is becoming prohibitive due to the lack of scalability of such systems, however fast they are. In fact, as the current literature demonstrates, see for example [8], the response time of access structures for metric spaces is linearly increasing with the size of the searched file.

⋆ This work was partially supported VICE project (Virtual Communities for Education), funded by the Italian government, and by DELOS NoE, funded by the European Commission under FP6 (Sixth Framework Programme).

The approach proposed in this paper is to use a Peer–to–Peer (P2P) structure composed of a network of nodes whose number can vary on the basis of the size of the data-set. The aim is to maintain the global response time stable as the data-set size grows. In this respect, the P2P paradigm is quickly gaining in popularity due to their scalability and self-organizing nature, forming bases for building large-scale similarity search indexes at low costs. However, most of the numerous P2P search techniques proposed in the recent years have focused on the single-key retrieval [7,11,13].

In particular, we present a distributed storage structure for similarity search in metric spaces that is based on the original idea of the *Content–Addressable Network* (CAN) [11], which is a distributed hash table abstraction over the Cartesian space. Our distributed storage structure, called *MCAN*, is able to index objects of a generic metric space. The advantage of the metric space approach to the data searching is its "extensibility", since in this way, we are able to perform the *exact match*, *range*, and *similarity* queries on any collection of metric objects. More in general, our proposal can be seen as a Scalable and Distributed Data Structure, (SDDS) – original proposal LH* [10] is intended for the primary key retrieval – which uses the P2P paradigm for the communication in a Grid-like computing infrastructure. A fundamental property of this paradigm is that insertion of an object, even if it implies a node split, does not require immediate update propagation to all network nodes.

The rest of the paper is organized as follows. In Section 2, we summarize the necessary background information. Section 3 presents the MCAN distributed structure and its functionality. Section 4 reports the results of performance evaluation experiments. Section 5 concludes the paper and outlines directions for future work.

2 Background

2.1 Content–Addressable Network (CAN)

The CAN is a distributed hash table that uses a function for mapping "keys" onto "values" in order to assign them a position in the table. In the CAN, the table is composed of a finite set of individual network nodes. Each node of the network is dynamically associated with a partition of a d-dimensional Cartesian space. Usually, the Cartesian space is a d-torus (in the sense that the coordinate space wraps), and is targeted to store (K, V) pairs, where K is an identifer of the object and V is a pointer to a copy of the object. The basic operations of the CAN are insertion, lookup and deletion of respective (K, V) pairs. In order to be compatible with the metric space, we generically refer to these pairs as "objects", and we use the notation $X \in \mathcal{S}$, for indicating an object X of an arbitrary space $\mathcal{S}$ of all possible pairs (or objects) $X \equiv (K, V)$.

From the formal point of view, we can define the mapping function of the CAN as:

$$G : \mathcal{S} \rightarrow P^N, \tag{1}$$

where P^N is an hyper–rectangle of $\mathbb{R}^N$ defined as:

$$P^N = [0, D_1] \times [0, D_2] \times \ldots \times [0, D_N], \tag{2}$$

with D_i denoting the $i - th$ side length of the CAN structure.

The principle of the CAN is to divide the hyper-rectangle P^N in a finite number of distinct rectangular zones, each of them associated to one and only one node of the network. The nodes are responsible for storing and searching of objects covered by their zone. Moreover, each node is aware of the nodes that cover adjacent zones, i.e., its neighbors. More precisely, for an N-dimensional space, two zones are neighbors if their sides overlap along $N-1$ dimensions and are adjacent along one dimension.

The basic operation in CAN is a lookup(key) function, which returns the corresponding "value" (the IP address of the node, for instance) for the given "key" (the coordinates of the point). This is useful for insertion, deletion, and retrieval purposes. The search starts from an arbitrary node of the CAN structure, and proceeds by routing a message towards its destination by simple greedy forwarding to the neighbor with coordinates closest to the destination coordinates. In general, if we divide the P^N uniformly in n zones, each node maintains $2N$ neighbors. Furthermore, the average routing path length is given by $(N/4)n^{(1/N)}$.

2.2 Metric Spaces

The mathematical metric space is a pair $\mathcal{M} = (\mathcal{D}, d)$, where $\mathcal{D}$ is the *domain* of objects and d is the *distance function* able to compute distances between any pair of objects from $\mathcal{D}$. It is typically assumed that the smaller the distance, the closer or more *similar* the objects are. For any distinct objects $X, Y, Z \in \mathcal{D}$, the distance must satisfy the following properties:

$d(X,X) = 0$	*reflexivity*
$d(X,Y) > 0$	*strict positiveness*
$d(X,Y) = d(Y,X)$	*symmetry*
$d(X,Y) \leq d(X,Z) + d(Z,Y)$	*triangle inequality*

2.3 Pivot-Based Filtering

In general, the pivot-based algorithms can be viewed as a mapping F from the original metric space $\mathcal{M} = (\mathcal{D}, d)$ to a N-dimensional vector space with the L_∞ distance. The mapping assumes a set $T = \{P_1, P_2, \dots P_N\}$ of objects from $\mathcal{D}$, called pivots, and for each database object O, the mapping determines its characteristic (feature) vector as $F(O) = (d(O,P_1), d(O,P_2), \dots d(O,P_N))$. We obtain a new metric space as $\mathcal{M}_N(\mathbb{R}^N, d^\infty)$. At search time, we compute for a query object Q the query feature vector $F(Q) = (d(Q,P_1), d(Q,P_2), \dots d(Q,P_N))$ and discard for the search radius r an object O if

$$d^\infty(F(O), F(Q)) > r \tag{3}$$

In other words, the object O can be discarded if for some pivot P_i,

$$\mid d(Q,P_i) - d(O,P_i) \mid > r \tag{4}$$

Due to the triangle inequality, the mapping F is *contractive*, that is all discarded objects do not belong to the result set. However, some not-discarded objects may not be relevant and must be verified through the original distance function $d(\cdot)$. For more details, see for example [4].

3 MCAN

The basic idea of our approach is to extend the CAN architecture in order to manage objects X of a generic metric space $\mathcal{M} = (\mathcal{D}, d)$. However, in metric spaces it is not possible to exploit any knowledge of coordinate information, and only distances between objects can be computed. To cope with this problem, we use the pivots paradigm for mapping the objects of the metric space to an N dimensional vector space. In particular, let $P_1, \ldots, P_N$ be the number of pivots selected from the metric data-set, we map an object $O \in D$, by means of the function $F()$ (introduced in the previous section) defined as:

$$F(O) : \mathcal{D} \to \mathbb{R}^N \;= (d(O, P_1), d(O, P_2), \ldots d(O, P_N)) \tag{5}$$

This virtual coordinate space is used to store the object O in the MCAN structure, specifically in the node that owns the zone where the point $F(O)$ lies. Note that, the coordinate space of the MCAN is not Cartesian since a distance between two objects in MCAN is evaluated by means of the L^∞ distance (instead of the Euclidean distance). Routing in MCAN works in the same manner as for the original CAN structures. An MCAN node maintains a coordinate routing table that holds the IP address and virtual coordinate zones of each of its immediate neighbors in the coordinate space.

3.1 Notation

In this section we provide a number of definitions required to present our results. We use the capital letter for indicating metric space objects $X \in \mathcal{D}$, the overline small letter for denoting the corresponding vector in the coordinate space $\overline{x} \in \mathbb{R}^N$, and x_i for representing the values of its i-th coordinate. Moreover, we denote a node of MCAN by the bold symbol **n**. Since there is no possibility of confusion, we use the same symbol $d(.)$ for indicating the distance between metric objects and for indicating the L^∞ distance between the corresponding point in the coordinate space, e.g., $d(\overline{x}, \overline{y}) = d^\infty(F(X), F(Y))$, where $\overline{x} = F(X)$ and $\overline{y} = F(Y)$. As we already explained, the MCAN is contractive, therefore $d(\overline{x}, \overline{y}) \leq d(X, Y)$ always holds.

Each node **n** maintains its region information referred as $\mathbf{n}.R$. Moreover, since the region $\mathbf{n}.R$ is an hyper–rectangle it can be uniquely identified by its vertex closer to the origin, denoted as $\mathbf{n}.R.\overline{x} = (\mathbf{n}.R.x_1, \mathbf{n}.R.x_2, \ldots, \mathbf{n}.R.x_N)$, and by the lengths of the relative sides, i.e., $\mathbf{n}.R.l_1, \mathbf{n}.R.l_2, \ldots, \mathbf{n}.R.l_N$. More precisely, the region $\mathbf{n}.R$ is defined as follows

$$\mathbf{n}.R = \{\forall \overline{x} \in \mathbb{R}^N \mid \forall\, i,\ \mathbf{n}.x_i \leq x_i < \mathbf{n}.x_i + \mathbf{n}.l_i\}$$

The node **n** also maintains the set of the neighbor nodes' information $\mathbf{n}.M = \{\mathbf{m}_1, \ldots, \mathbf{m}_h\}$.

Given a point $\overline{x} = F(X)$, the predicate $X \in \mathbf{n}$ allows us to check if the corresponding point $\overline{x}$ lies is in the zone maintained by the node **n**. More formally:

$$X \in \mathbf{n} \;\Leftrightarrow\; \overline{x} \in \mathbf{n}.R$$

A range query of radius r and centered in the object C is denoted as $\mathcal{Q} = (\overline{c}, r)$. The predicate $\mathcal{Q} \cap \mathbf{n}$ allows to check if the query region $\mathcal{Q}$ intersects the zone associated with **n**. Note that, the range query in the L^∞ space is given by an hypercube of side $2r$ centered in $\overline{c}$.

We can now introduce the formal definition of an N-dimensional MCAN structure, referred as MCAN^N, which is composed of a set of k ($k > 0$) network nodes $\{\mathbf{n}_1, \ldots, \mathbf{n}_k\}$ such as:

1. $\forall\, i, j \mid i \neq j \;\; \mathbf{n}_i.R \cap \mathbf{n}_j.R = \emptyset$
2. $P^N = \bigcup_{i=1}^{k} \mathbf{n}_i.R$
3. $\mathbf{n} \in \mathbf{m}.M \Leftrightarrow$
   $\exists k \mid 1 \leq k \leq N, (\mathbf{n}.R.x_k + \mathbf{n}.R.l_k = \mathbf{m}.R.x_k) \vee (\mathbf{m}.R.x_k + \mathbf{m}.R.l_k = \mathbf{n}.R.x_k),$
   $\forall w \neq k\; [\mathbf{n}.R.x_w, \mathbf{n}.R.x_w + \mathbf{n}.R.l_w[\; \cap [\mathbf{m}.R.x_w, \mathbf{m}.R.x_w + \mathbf{m}.R.l_w[\; \neq \emptyset$

In the definition, Point 1. states that the zones covered by the network nodes do not overlap. Point 2. states that the union of the zones cover the whole MCAN^N space P^N (there are no holes). Finally, Point 3. declares the condition for a network node **n** to be a neighbor of **m** (as explained in Section 2).

3.2 Construction

An important feature of the CAN structures is its capability to dynamically adapt to data-set size changes. As we will see in the experimental evaluation, we are interested in preserving the scalability of the MCAN, which means that we want to maintain stable the response time of the queries. Since the size of the space allocated to store objects in each node is limited, when a node exceeds its limit it splits by sending a subset of its objects to a free node and by assigning its part of original region. Note that, limiting the storage space, and then the number of objects each node can maintain, we limit also the number of distance computations a node have to evaluate during a range query computation.

It is important to observe that in some cases we might want to use all the nodes available in the network. Previous work like [11] have studied this possibility in a generic CAN structure by allowing a node to split even if it does not exceed it storage space. Obviously, such methodology can also be applied in our MCAN. On the other hand, in a P2P environment, we would like to let the nodes the possibility to freely join and leave the network, without damaging it. As explained in [11], this is possible with a CAN, which even provides some fault-tolerance capabilities [12].

Since the pivots needed to be determined before the insertion starts, we assume a characteristic subset of the indexed data-set (about 500 objects) is known at the beginning. In the MCAN, we use the Incremental Selection algorithm described in [4].

3.3 Insertion

An insert operation can start in any node of the MCAN. It starts by mapping the inserted object X to the virtual coordinate space using function $F()$, then it checks if $\overline{x} = F(X)$ lies in the zone maintained by the node **n** itself (i.e. $X \in$ **n**). If this is not the case, the node has to forward the insertion request. From this point, the insertion proceeds with the greedy routing algorithm used for standard CAN structures: the inserting node forwards the insertion operation to the neighbor node which is closer to the point $\overline{x}$ by using the L^{∞} distance. The objective is to find the node **n** for which $X \in$ **n**, minimizing the number of messages. If $\overline{x}$ lies in the region maintained by the receiving node, the object X is stored there, otherwise a neighbor node is selected with the same technique and the insert operation is forwarded again until the object X is inserted.

The node **m** which stores the object X must reply to the node who started the insert operation. If the node **m** exceeds its capacity it is split. Eventually, the object X is inserted in **m** or in the new allocated node.

3.4 Split

In MCAN, we apply a balanced split, that is the resulting regions contain practically the same amount of data (object occupancy). During this process, the splitting node will just request a node from a free node list to join the network, and one half of the data, in terms of occupancy, is reallocated there.

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

$$\mathbf{n}_1.R' \cup \mathbf{n}_2.R = \mathbf{n}_1.R \ , \ \ \mathbf{n}_1.R' \cap \mathbf{n}_2.R = 0$$

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

$$\mathbf{n}_1.R'.x_s = \mathbf{n}_1.R.x_s, \ \mathbf{n}_2.R.x_s = \mathbf{n}_1.R.x_s + \mathbf{n}_1.R'.l_s, \ \ \mathbf{n}_2.R.l_s = \mathbf{n}_1.R.l_s - \mathbf{n}_1.R'.l_s$$

Note that that we only have to chose s and $\mathbf{n}_1.R'.l_s$. In order to decide s, for each dimension i we find $\mathbf{n}_1.R'.l_i$ that divide the objects into two halves. To avoid regions with small sides we chose s as the dimension i for which $|\mathbf{n}_1.R'.l_i - \mathbf{n}_1.R.l_i/2|$ is minimum.

After the splitting process, the node $\mathbf{n}_1$ sends a message to all its neighbors informing them about the update of its region. To those neighbors, which are also neighbors of $\mathbf{n}_2$, it sends also information about the new node. The new node is informed by $\mathbf{n}_1$ about its neighbors that are a subset of the $\mathbf{n}_1$ neighbors. At the end, $\mathbf{n}_1$ can discard information about the nodes that no more are its neighbors.

3.5 Range Query

A range query operation can start at any MCAN node. As shown in Figure 1, for a given query object and range radius, there is a certain number of nodes whose regions intersect the query region (which is an hypercube $\mathcal{Q} = (\overline{c}, r)$ as defined in Section 3.1). Obviously, only the intersecting nodes must process the range query operation. The requesting node maps the query object in the virtual coordinate space using the function $F()$. Then it checks if it is involved in the range query operation (when it intersects $\mathcal{Q}$). If the node is not involved in the query, it forwards the range query operation to the neighbor node that is closest to region $\mathcal{Q}$, using the L^{∞} distance. This operation is performed in a similar way as described for the insert operation.

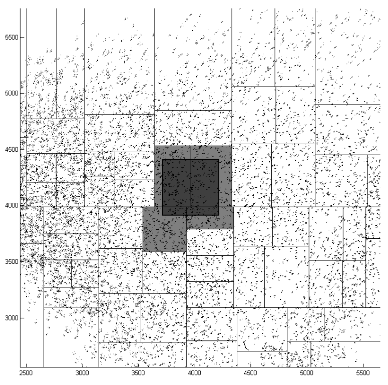

Fig. 1. Example of range query in a two dimensional space. The darker square is the query region, while the brighter rectangles correspond to the involved nodes.

When a node that is involved in the range query is reached by the query request, it forwards it to each neighbor that is also involved and then it starts processing the range query over its local data-set. During the range query execution inside a single node, a local access structure can also be used. In this paper, we used the same pivots chosen to define the MCAN space to reduce the number of distance evaluations performed inside a single node. Using the pivot-based filtering, we are able to significantly reduce the number of distance evaluations inside the nodes. In a more sophisticated implementation of MCAN, each node could have its own local data structure to efficiently search inside a single node.

In order to allow the requesting node to know when all the nodes involved in the query have finished to work, the MCAN proceeds as follows:

- A node involved in the range query:
  - it receives the range query request $\mathcal{Q} = (\bar{c}, r)$ and a (possibly empty) list of the nodes already involved in the query (which we refer to as *INL*),
  - it forwards the query request to its neighbors involved in the query which are not included in *INL* adding them to the list and sending the new *INL** to them,
  - it computes the query over its local data,
  - it replies with a message containing its result set (if any) and the *INL**.
- The requesting node as it receives the reply messages, updates a local list of the involved nodes in the query, and marks the ones that have already answered.
- The requesting node will know that the operation is terminated when all nodes of the local list will have replied (i.e., when all the nodes of the list have been marked). The result set of the query is given by union of the result sets of the replying nodes.

Note that, the first node involved in the query receives an empty *INL*. Another important observation is that with this scheme we do not guarantee that a node does not receive multiple requests for the same query. However, this is not a problem since each distinct query is associated with a unique identifier, so that a node ignores multiple requests.

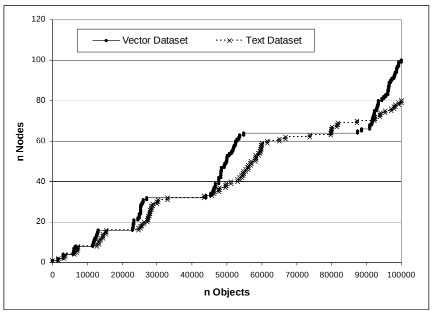

Fig. 2. Number of nodes for increasing data-set size.

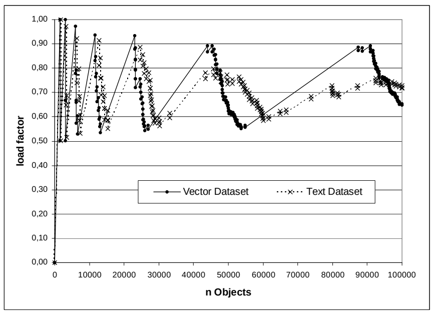

Fig. 3. Average number of objects per node for the MCAN^3 and for increasing data-set size.

4 Performance Evaluation

In this section we present a preliminary experimental evaluation of MCAN. The metric data-sets used are: 100,000 of 45-dimensional vectors of color features extracted from images; 100,000 Czech sentences of length between 20 and 300 characters. Vectors are compared by the Euclidean distance measure while for sentences we use the Edit distance.

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

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

In Figure 2, we report the number of nodes in the system as the data-set grows, for the MCAN^3 case (the other cases are very similar). Note from these experiments that, the number of nodes exhibit a stepwise behavior. This is due to the fact that the objects are randomly ordered, therefore the nodes are filled uniformly and then they tend to split at the same time. This is particularly evident for the vector data-set, where the objects have a fixed size, while the size of objects of the text data-set (strings) is variable.

In Figure 3, we report the average load factor for both data-sets. We define the load factor as the total number of objects stored in the MCAN structure divided by the capacity of storage available on all nodes. As can be seen in the figure, the values are always between 0.5 and 1. This is always guaranteed, because when a node is split, half

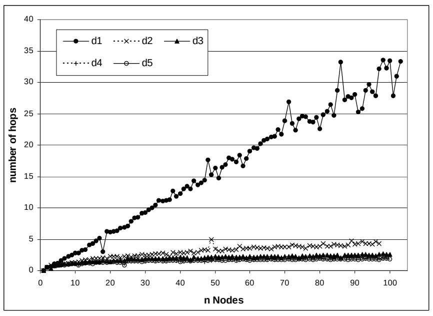

Fig. 4. Average number of hops for different dimensions as the number of nodes grows (vector data-set).

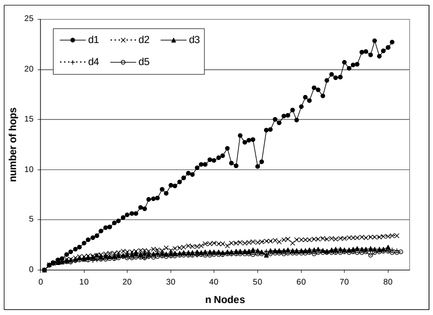

Fig. 5. Average number of hops for different dimensions as the number of nodes grows (text data-set).

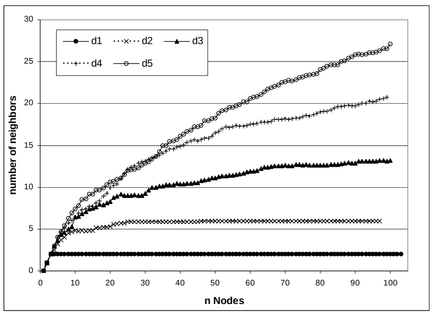

Fig. 6. Average number of neighbors for different dimensions as the number of nodes grows (vector data-set).

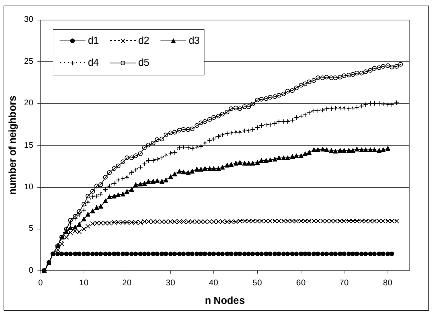

Fig. 7. Average number of neighbors for different dimensions as the number of nodes grows.

of the objects are migrated to the new node, therefore the node occupation cannot be less than 50%.

In Figures 4 and 5, we report the average number of hops an insert operation travels, starting form a random node. The number of hops is strictly correlated to the average number of neighbors each node has: the more the neighbors, the less the hops. In Figures 6 and 7, we report the average number of neighbors as a function of the total number of nodes for different space dimensionality. Comparing the number of hops with the number of neighbors, we can see that a good choice for the space dimensionality could be $N = 3$. In fact, by using more than 3 dimensions we do not reduce significantly the number of hops but significantly increase the number of neighbors and correspondingly the complexity of choosing next node in forwarding.

4.1 Range Query

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

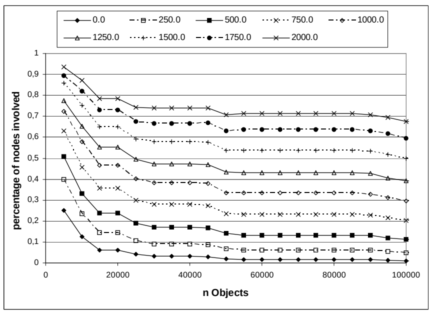

Fig. 8. Percentage of nodes involved in the range query as function of the data-set size for different radii (vector data-set).

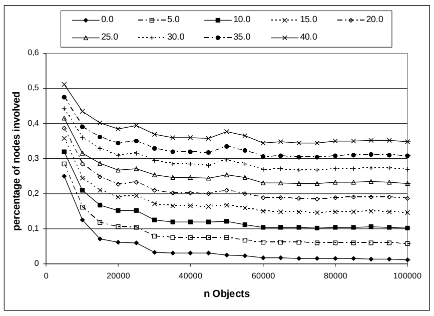

Fig. 9. Percentage of nodes involved in the range query as function of the data-set size for different radii (text data-set).

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

In Figures 8 and 9 we report the average percentage of nodes involved during a range query operation for different radii as the data-set size grows. Observe that the bigger is the radius of the range query, the more are the nodes involved in the query evaluation. In a naive distributed system we could randomly distribute the objects among the nodes but in this case we would always involve all the nodes even for small radii.

For simple operations like the exact match, the standard CAN has been proved to be scalable. MCAN extends CAN by allowing similarity operations over generic metric space data-sets. In this scenario, we must be able to perform more complex operations such as similarity range queries. To preserve scalability also for such operations, we need more nodes as the complexity of the query grows. This aspect is evident in the plot of Figures 8 and 9, where the percentage of nodes involved for a small radius is smaller then the ones we obtains for greater radii. Note that, for a given range query, the percentage of nodes involved is almost constant. In fact, for a given range query the number of results is linearly dependent on the number of objects in the data-set and then the number of nodes involved is proportional to the number of results.

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

In Figures 10 and 11, we report the average complexity of the range query operations as function of the number of equivalent distance computations of the most stressed node. This quantity measures in a way the *intraquery parallelism* as the parallel re-

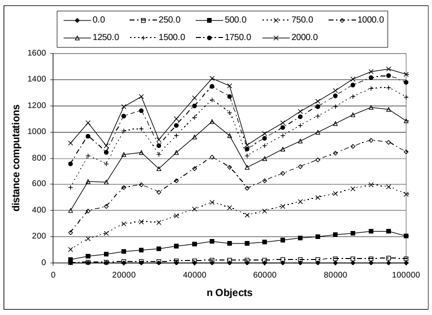

Fig. 10. Average number of distances evaluated by the most stressed node for each query and for different query range (vector data-set).

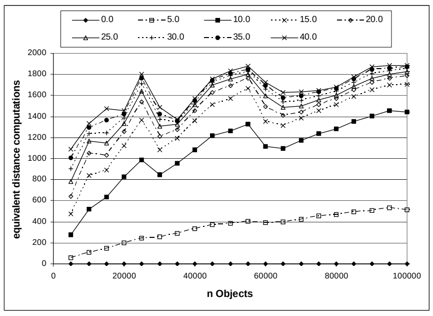

Fig. 11. Average number of equivalent distances evaluated by the most stressed node for each query and for different query range (text data-set).

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

5 Related Work and Conclusions

There have been several recent attempts to propose distributed structures for multi-dimensional or vector-based data. The MAAN structure [5] use locality preserving hashing to support multi-attribute and range queries under the Chord protocol. The kd-trees and space-filling curves have been used by Prasanna, Yang and Garcia-Molina in [9] to support multi-dimensional range queries in P2P environments. A P2P system for information retrieval based on the vector space model and the latent semantic indexing together with the CAN P2P protocol has been proposed by Tang, Xu and Dwarkadas [14]. The problem of vector-based similarity search in P2P Data Networks has nicely been formalized by Banaei-Kashani and Shahabi [1] as the family of Small-World based Access Methods, SWAM. So far, the only native metric-based distributed data structure is the GHT* [2,3].

To the best of our knowledge, the MCAN structure is the first attempt to bridge Content–Addressable Networks and the capabilities of metric space indexing. MCAN is based on the concept of choosing pivots to map objects of a generic metric space in a multidimensional vector space of the MCAN. Since the mapping is contractive, 100% recall for queries processed by the MCAN is guaranteed.

The results summarized in Figures 8 and 9 should be considered as the first attempt to also demonstrate the *interquery parallelism* ability of MCAN. In fact, if on the one hand it is important to guarantee fast response time to individual queries, on the other hand the query should not involve the whole network, because other queries can be issued in the network at the same time, and not active nodes can simultaneously start evaluating them. Obviously, queries with large radii need more computational resources than small queries, but typically, there is always sufficient space for other queries to run. Also observe that the computational load on nodes activated by a query is not the same and on some nodes the load is really minor. We are planning to fully investigate this issue in the near future.

Further future directions include the implementation of the Nearest Neighbor queries, more sophisticated leaving and join policies, and transaction management.

References

1. F. Banaei-Kashani and C. Shahabi. Swam: a family of access methods for similarity-search in peer-to-peer data networks. In *CIKM '04: Proceedings of the Thirteenth ACM conference on Information and knowledge management*, pages 304–313. ACM Press, 2004.
2. M. Batko, C. Gennaro, and P. Zezula. A scalable nearest neighbor search in p2p systems. In *Proc. of the the 2nd International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, Lecture Notes in Computer Science, August 2004. To appear.
3. M. Batko, C. Gennaro, and P. Zezula. Scalable similarity search in metric spaces. In *Proc. of the DELOS Workshop on Digital Library Architectures: Peer-to-Peer, Grid, and Service-Orientation*, pages 213–224, June 2004.
4. B. Bustos, G. Navarro, and E. Chvez. Pivot selection techniques for proximity searching in metric spaces. In *Proc. of SCCC01*, pages 33–40, 2001.
5. M. Cai, M. Frank, J. Chen, and P. Szekely. Maan: A multi-attribute addressable network for grid information services. In *GRID '03: Proceedings of the Fourth International Workshop on Grid Computing*, page 184, Washington, DC, USA, 2003. IEEE Computer Society.
6. E. Chvez, G. Navarro, R. Baeza-Yates, and J. L. Marroqun. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.
7. R. Devine. Design and implementation of DDH: A distributed dynamic hashing algorithm. In *FODO*, pages 101–114, 1993.
8. V. Dohnal, C. Gennaro, P. Savino, and P. Zezula. D-index: Distance searching index for metric data sets. *Multimedia Tools and Applications*, 21(1):9–13, 2003.
9. P. Ganesan, B. Yang, and H. Garcia-Molina. One torus to rule them all: multi-dimensional queries in p2p systems. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 19–24, New York, NY, USA, 2004. ACM Press.
10. W. Litwin, M.-A. Neimat, and D. A. Schneider. LH* — a scalable, distributed data structure. *ACM Transactions on Database Systems*, 21(4):480–525, 1996.
11. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proc. of ACM SIGCOMM 2001*, pages 161–172, 2001.
12. J. Saia, A. Fiat, S. D. Gribble, A. R. Karlin, and S. Saroiu. Dynamically fault-tolerant content addressable networks. In *IPTPS*, pages 270–279, 2002.
13. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proc. of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.
14. C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks, 2002.