A Content-Addressable Network for Similarity Join in Metric Spaces [*]

Claudio Gennaro
ISTI - CNR
Pisa - Italy
gennaro@isti.cnr.it

ABSTRACT

Similarity join is an interesting complement of the well-established similarity range and nearest neighbors search primitives in metric spaces.

However, the quadratic computational complexity of similarity join prevents from applications on large data collections. We present $MCAN^+$, an extension of $MCAN$ (a Content-Addressable Network for metric objects) to support similarity self join queries. The challenge of the proposed approach is to address the problem of the intrinsic quadratic complexity of similarity joins, with the aim of limiting the elaboration time, by involving an increasing number of computational nodes as the dataset size grows. To test the scalability of $MCAN^+$, we used a real-life dataset of color features extracted from one million images of the Flickr photo sharing website.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process

Keywords

Similarity Join, Content-Addressable Network, Metric Space

1. INTRODUCTION

Similarity join is a database primitive that finds all pairs of records within a predefined distance threshold of each other. The similarity join has been successfully applied to a large class of applications, data analysis, data mining, location-based applications, and time-series analysis. This search paradigm has recently been generalized into a model in which a set of objects can only be pair-wise compared through a distance measure d satisfying the *metric space* properties (i.e, the positivity, symmetry, and triangle inequality) [4].

The problem of similarity join emerges naturally in a variety of applications where the user is not only interested in the properties of single data objects but also in the properties of the data set as a whole, as, for instance, in data mining applications. Considering the typical clustering task of data mining, many of the state-of-the-art algorithms require to process all pairs of data items which have a distance not exceeding a user-given parameter ϵ. Consequently, many data mining algorithms can be directly performed on top of a similarity join as proposed in [3].

However, the quadratic computational complexity of similarity joins prevents from applications on large data collections. To give an idea, let us consider a database of one million records. For computing the similarity self join, we need a number of distance evaluations of the order of one thousand billions. This is the main focus of our paper, in which we extend the existing metric distributed data structure, $MCAN$ [6, 7], to efficiently support similarity self join searches.

The remainder of our paper is organized as follows: In Section 2, we introduce and discuss our new approach to similarity self join based on the extension of the distributed data structure $MCAN$. The experimental evaluation of our approach is presented in Section 3. Section 4 concludes the article.

2. THE MCAN+

The $MCAN^+$ is an extension of the $MCAN$ distributed data structure [6, 7] a Content-Addressable Network [8] for metric space objects.

Since in metric spaces only distance among objects is known and it is not possible to exploit any knowledge of coordinate information, we use the pivot paradigm for projecting objects of the metric space into n-dimensional vectors (please refer to [6] for details).

We use the lower case letter for indicating a metric space objects $x \in \mathcal{D}$, the overlined small letter for denoting its corresponding vector in the coordinate space $\overline{x} = F(x) \in \mathbb{R}^n$. Moreover, we denote a peer of $MCAN$ by the bold symbol $\mathbf{p}$. Since there is no possibility of confusion, we use the same symbol $d(.)$ for indicating the distance between metric objects and for indicating the d^∞ distance between the corresponding points in the coordinate space, e.g., $d(\overline{x}, \overline{y}) = d^\infty(F(x), F(y))$, where $\overline{x} = F(x)$ and $\overline{y} = F(y)$. It is impor-

[*]This work was partially supported by the SAPIR (Search In Audio Visual Content Using Peer-to-Peer IR) project, funded by the European Commission under IST FP6 (Sixth Framework Programme) and by the NeP4B project (Networked Peers for Business), funded by the Italian government.

INFOSCALE 2008, June 4-6, Vico Equense, Italy
Copyright © 2008 978-963-9799-28-8
DOI 10.4108/ICST.INFOSCALE2008.3526

tant to note that, the distance $d(\overline{x}, \overline{y})$ is contractive, therefore $d(\overline{x}, \overline{y}) \leq d(x, y)$ always holds.

Each peer $\mathbf{p}$ maintains its region (a hyper-rectangle) information referred as $\mathbf{p}.R$ and stores all objects x such that $\overline{x} \in \mathbf{p}.R$. The peer $\mathbf{p}$ also maintains the set of the neighbor peers' information $\mathbf{p}.M = \{\mathbf{m}_1, \ldots, \mathbf{m}_h\}$. Moreover, during the creation of the structure of *MCAN*, we assign a unique numeric identifier i with each peer, which we denote as $\mathbf{p}.id$. While $\mathbf{p}(i)$ denotes the peer corresponding to the identifier i.

The *similarity join* is a search primitive that combines objects of two sets $X = \{x_1, ..., x_N\}$ and $Y = \{y_1, ..., y_M\}$ into one set such that $X \overset{sim}{\bowtie} Y = \{\langle x_i, y_j\rangle \in X \times Y \mid d(x_i, y_j) \leq \epsilon\}$,. Where the threshold ϵ is a non negative real number. If the sets X and Y coincide, we talk about the *Similarity Self Join* (SSJ). in this article, we only concentrate on this version of similarity joins.

The idea behind the $MCAN^+$ is to enlarge by μ peers' bounding regions $\mathbf{p}.R$ equally in all directions so that they overlap their neighbors' regions. This principle ensures that there always exists a region of a peer for every qualifying pair $\langle x, y\rangle$ (with $d(x, y) \leq \mu$) where the pair occurs. However, unlike *MCAN* the peer maintains a bounding region that overlaps of μ the regions of its neighbors. The peers of $MCAN^+$ keep track of this region, which we refer to as *core region*, and denote by $\mathbf{p}.R$ (exactly as in *MCAN*). Moreover, we call the overlapping rectangle *extended region* and we denote it by the symbol $\mathbf{p}.E$. As explained above, since in $MCAN^+$ the regions may overlap, some objects are replicated on more peers. As it easy to understand, the greater is μ the greater is the replication. This aspect implies that the insertion algorithm of $MCAN^+$ is more sophisticated than the one of standard CAN structures, as explained in the next section.

2.1 Similarity Self Join Algorithm

The outline of the SSJ algorithm is as follows: each peer executes the join query (on its subset) independently. The partial results from all peers are then concatenated and form the final answer. Note that, since $MCAN^+$ works in a contractive (d^∞) space, when we find a pairs of objects $\langle x, y\rangle$ for which $d(\overline{x}, \overline{y}) \leq \epsilon$, in order to know if the pair belongs to the result, we must also check if $d(x, y) \leq \epsilon$.

A naive implementation of SSJ could simply exploit the sliding window algorithm introduced in [5] on the local subset of objects maintained by the peer. Note that, the sliding window requires the objects to be ordered with respect to an extra pivot t, typically (but not necessarily) distinct from the ones used for creating the $MCAN^+$ structure. However, an important issue arises in the application of this simple algorithm: the problem of duplicate pairs in the result of the join query. This fact is caused by the copies of objects which are stored into distinct peers. Precisely, during the construction of $MCAN^+$ we append extra information with each indexed object. This information keeps track of peers that maintain a copy of the object itself. In particular, each metric object x of $MCAN^+$ has the following two attributes:

- $x.own$: it stores the *id* of the peer that holds the object x in the core region (there can be only one of such a peer, since core regions do not overlap). We refer the peer of $id = x.own$ as *owner* of x.

- $x.rep$: it stores a boolean flag telling us if the objects x is replicated on more than one peer (true) or not (false).

$x.own$	$x.rep$	region
$= k$	*false*	A
$\neq k$	*false*	–
$= k$	*true*	B
$\neq k$	*true*	C

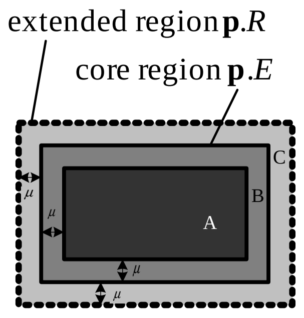

Figure 1: Illustration of the various zones of a peer of $MCAN^+$.

Note that, when $x.rep = false$, the peer of $id = x.own$ exclusively owns x.

Algorithm 2.1 includes the function *SimilaritySelfJoin*, which takes the *id* k of a peer and the threshold ϵ as input parameters and returns the qualifying pairs of the peer i. We use a procedural approach to present $MCAN^+$ algorithms, by implicitly assuming that the parameters of functions and procedures are sent via a message-passing interface. Algorithm 2.1 simply starts invoking the function *SimilaritySelfJoin* on all peers of $MCAN^+$, and collects the results coming from them. Function *SimilaritySelfJoin* invokes in turn the *SlidingWindowSearch* function, which takes as input the threshold ϵ and returns the set of the pairs that potentially belong to the result set and which have to be checked using the metric distance d.

However, before evaluating the distance of each pair returned, it assesses if other peers have a replica of it. To achieve this task, *SimilaritySelfJoin* exploits the attributes *own* and *rep* described above.

To better understand the behavior of *SimilaritySelfJoin*, please see Figure 1, which illustrates the zones of a peer where an object can lay. The innermost rectangular zone A, is the portion of the core region $\mathbf{p}.R$ that does not overlap with other peer regions. The surrounding zone B corresponds to core region minus A, i.e., $B = \mathbf{p}.R - A$, and C the extended region minus the core region, i.e., $C = \mathbf{p}.E - \mathbf{p}.R$. We can infer the position of an object x in a peer, with respect those zones, by examining its attributes *own* and *rep.*, The principle of the algorithm is simple if we observe that the problem of replication occurs when the objects of a pair are owned by distinct peers, e.g., $\overline{x} \in B$ and $\overline{y} \in C$. The idea here is to exploit the *id*s of the peers to decide which one must consider the pair, for instance, by allowing the peer with the lowest *id* to consider the pair (however, any other determinist scheme based on *id*s would work as well).

More specifically, let x and y be two objects of a pair returned by *SlidingWindowSearch* function, and let k the *id* of the peer $\mathbf{p}$ currently executing the algorithm. In order to avoid duplicates the algorithm exploits the knowledge of the attributes of x and y, as in the following:

1. If both $\overline{x}$ and $\overline{y}$ are in zone A, the pair $\langle x, y\rangle$ is not replicated and will be considered (first *if* statement of *SlidingWindowSearch* function).

2. If $\overline{x} \in B$ ($\overline{y} \in B$) and $\overline{y} \in B \bigcup C$ ($\overline{x} \in B \bigcup C$), then there is a chance that the pair $\langle x, y\rangle$ is also reported by another peer. If also $\overline{y} \in B$ ($\overline{x} \in B$), the pair is considered. If instead $\overline{y} \in C$ ($\overline{x} \in C$), then $y.id \neq x.id$. We allow only the peer with lowest *id* to considered the pair (i.e., $id = min(x.id, y.id)$), and force the other peer to ignore the pair. The second and the third *if* statements of *SlidingWindowSearch* function accomplish this part of the algorithm.

3. If none of these conditions occur, we ignore the pair $\langle x, y\rangle$. In fact, in this case, it can be $\overline{x} \in A$ and $\overline{y} \in C$ (or vice versa $\overline{y} \in A$ and $\overline{x} \in C$), therefore the distance $d(x, y) \geq d(\overline{x}, \overline{y}) > \mu \geq \epsilon$. Otherwise, both $\overline{x}$ and $\overline{y}$ are in C, therefore the pair will be considered by another peer.

Since all peers of $MCAN^+$ respect this same scheme, there is no risk to produce duplicate pairs.

Algorithm 2.1. *Similarity Self Join*

```
S := ∅;
for each p ∈ MCAN⁺
    S := S + SimilaritySelfJoin(p.id, ε)
end for each

function SimilaritySelfJoin(k, ε): set
    P := SlidingWindowSearch(ε);
    R := ∅;
    for each ⟨x, y⟩ ∈ P
        CheckDist := false;
        if (not x.rep) and (not y.rep) then
            # both x̄ and ȳ are in zone A
            CheckDist := true;
        end if
        if x.own = k and x.rep then
            # x̄ is in zone B
            if y.own ≥ k then
                # ȳ is in zone B⋃C
                CheckDist := true;
            end if
        end if
        if y.own = k and y.rep then
            # ȳ is in zone B
            if x.own ≥ k then
                # x̄ is in zone B⋃C
                CheckDist := true;
            end if
        end if
        if CheckDist then
            if d(x, y) ≤ ε then
                R := R + ⟨x, y⟩;
            end if
        end if
    end for each
    return R;
end function
```

2.2 Insertion Algorithm

In order to exploit the SSJ algorithm presented in the previous section, $MCAN^+$ must employ an insertion algorithm more sophisticated than the one used in $MCAN$. The insertion operation can start from any peer $\mathbf{p}$ of the $MCAN^+$, and initiates by mapping the object x to insert into the virtual coordinate space using function $F()$. Then, if $\overline{x} = F(x) \in \mathbf{p}.R$ x is stored in $\mathbf{p}$. On the contrary, if $\overline{x} \notin \mathbf{p}.R$ the peer must forward the insertion request to its neighbor peer closer to the point $\overline{x}$. The objective is to find the peer $\mathbf{m}$ for which $\overline{x} \in \mathbf{m}.R$, minimizing the number of messages.

So far, the insertion algorithm works exactly as in $MCAN$. However, after this preliminary phase the peer that stores the object must start a second phase that implements the replication principle of $MCAN^+$, as described in Algorithm 2.2. The algorithm includes procedure *Insert* and function *Replicate*. Procedure *Insert* accomplishes the first phase of the insertion operation and has two input parameters: the *id* i of the peer that takes care of the insertion and the object x to be inserted. The peer checks if the object belongs to its core region. If so, it sets $x.own$ to i, stores the object, and sends a copy of it to its neighbors (by mean of the *Replicate* function). If at least one of the neighbors informs the peer that the object x has been replicated, than it sets the attribute $x.rep$ to *true*. If the core region of the peer does not contain $\overline{x}$, the peer forwards x to the closest neighbor peer to the point $\overline{x}$, by recursively invoking the *Insert* procedure.

Replicate function accomplishes the second phase of the insertion operation, which deals with replication and has the same input parameters of *Insert* plus the *id* j of the sender (the function caller). The returned value indicates if the object has been replicated by the peer (true) or not (false). The receiving peer first checks if the object x belongs to its extended region (i.e., if $\overline{x} \in \mathbf{p}(i).E$), if so, it sets $x.rep$ to *true*, stores x (ignore for now the *if* statement), recursively invokes *Replicate* function on its neighbors, and, finally, returns *true* to the peer from which it received the object x. On the contrary if $\overline{x} \notin \mathbf{p}(i).E$, the peer simply does nothing and returns *false*. All the neighbor peers involved in the replication phase, performs the same algorithm. To guarantee algorithm termination, if a peer receives multiple invocations of *Replicate* for the same object x it ignores the following invocations. Moreover, a peer does not invoke *Replicate* more than once on the same peer for the same object x.

The reason for checking the condition $x.own > i$ (in function *Replicate*) before storing the object x, has to do with a mere optimization issue. We exploit the fact that we can discard a replicated object if it will be never tested for similarity during the SSJ. In fact, the condition for not storing x is that the object is replicated ($x.rep = true$) and that x is owned by another peer ($x.id \neq i$). This exactly corresponds to the case $\overline{x} \in C$ (see the table of the previous section). In this case, during the SSJ evaluation, as explained in the previous section, only the peer with the lowest *id* will consider this object when occurring in a pair. Therefore, the peers that have *id* i greater than $x.id$ can safely omit to store it. Exploiting this optimization we save more or less half space due to replication and some distance computations.

Algorithm 2.2. *Insertion*

```
procedure Insert(i, x)
    if x̄ ∈ p(i).R then
        x.own := i;
        Store(x);
        x.rep := false;
        for each m ∈ p(i).M
            IsReplicated := Replicate(m.id, x, i);
            if IsReplicated then
                x.rep := true;
            end if
        end for each
    else
        m := GetNearestNeighbor(p.M, x̄);
        Insert(m.id, x);
    end if
end procedure

function Replicate(i, x, j): boolean
    if x̄ ∈ p(i).E then
        x.rep := true;
        if x.own > i then
```

```
            Store(x);
        end if
        for each $m \in p(i).M$
            if $m$.id $\neq j$ then
                Replicate($m$.id, $x$, $i$);
            end if
        end for each
        return true;
    else
        return false;
    end if
end function
```

The advantage of keeping the two phases of the $MCAN^+$ construction (insertion and replication) distinct is that we can defer the replication phase indefinitely. In practice, $MCAN^+$ construction without replication is identical to an $MCAN$ or an $MCAN^+$ with $\mu = 0$. Whenever we need to deal with SSJ queries with $\epsilon > 0$, we can force all peers to start the replication phase for any $\mu \geq \epsilon$.

3. PERFORMANCE EVALUATION

In order to demonstrate the suitability of $MCAN^+$ to the problem of SSJ, we have conducted several experiments using a large real-life dataset of MPEG-7 Scalable Color Descriptors extracted from one million images of Flickr photo sharing website [2]. The distance used for this visual descriptor is the L_1, as suggested by the MPEG-7 standard [1].

The time complexity of the Nested Loop (NL) algorithm (i.e., the exhaustive search) is $\frac{N \cdot (N-1)}{2}$. However, to make a fair assessment of the magnitude of the SSJ problem, we first tested the sequential implementation of the Sliding Window algorithm on the Flickr dataset. The values of the thresholds ϵ produce a number of pairs that range from 0.00005% (for $\epsilon = 0$) to 0.0029% (for $\epsilon = 10$) of the five hundred billions of possible pairs. The cost in terms of distance computations grows linearly from about one billion for $\epsilon = 0$ to 21.7 billions for $\epsilon = 10$, which correspond to 15 minutes and 5 hours and 20 minutes (using a machine equipped with an Intel 2.13GHz processor), respectively.

Note that, since Sliding Window algorithm exploits the triangle inequality property of metric spaces, it produces a strong performance improvement with respect the NL. Indeed, consider that the time estimated for the simple NL algorithm is more than five days.

3.1 Scalability of MCAN$^+$

We analyze the behavior of a $MCAN^+$: involving 2 pivots for mapping the metric space in a 2–dimensional vector space and another pivot for employing the Sliding Window algorithm inside the peers. Since in this work we concentrate our attention on scalability issues, we fix the storage space available for each peer and then, starting from a single peer, we add objects into the system. When a peer reaches its storage space limit, it splits. We partition the dataset in 32 blocks of 31,250 objects (i.e., SC descriptors), and we double the dataset size until we reach the total size of 1,000,000 objects. Before inserting the dataset in $MCAN^+$, we have randomly mixed it to prevent influence of the order of images acquisition on the performance of scalability experiments. The objective of this research is to try to approach this problem by exploiting the parallelism of peer computations, in order to limit the parallel similarity join computation time. However, as explained above, when we double the problem size, the computational demand is quadruplicated. For this reason the number of computational

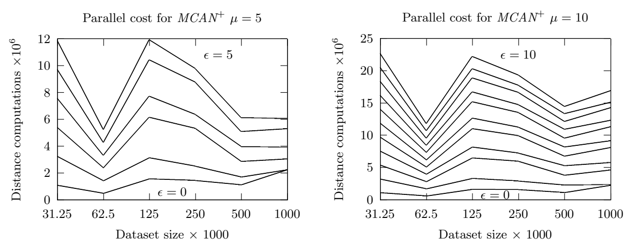

Figure 2: Parallel cost of two $MCAN^+$ settings ($\mu = 5$ and $\mu = 10$) for growing dataset for increasing values of ϵ.

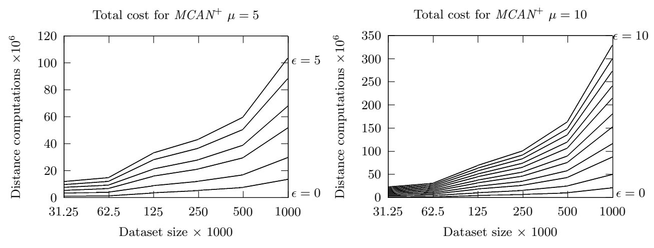

Figure 3: Total cost of two $MCAN^+$ settings ($\mu = 5$ and $\mu = 10$) for growing dataset for increasing values of ϵ.

peers is quadruplicated accordingly, as the following table shows:

Dataset Size ×1000	31.25	62.5	125	250	500	1,000
Number of Peers	1	4	16	64	256	1024

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

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

- *total distance computations* – the sum of the number of distance computations on all peers,
- *parallel distance computations* – the maximal number of distance computations among the local SSJ performed by peers in parallel.

To give an example, consider an $MCAN^+$ with just three peers executing a SSJ with number of distance computations being respectively 200, 300, and 500. In this case, the total distance computations would be 1,000 and the parallel distance computations would be 500.

As discussed above, an $MCAN^+$ with replication μ is able to perform SSJ with $\epsilon \leq \mu$. For this reason we study the performance of two different $MCAN^+$ settings: one with $\mu = 5$ and one with $\mu = 10$. We believe that a SSJ with $\epsilon > 10$ is

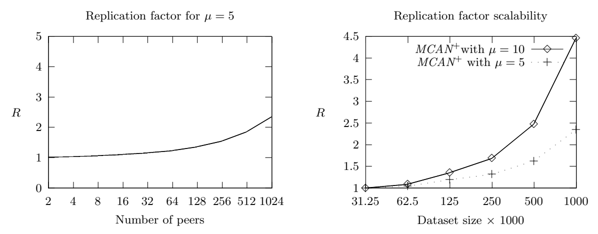

Figure 4: Replication factor R for different number of peers and with $\mu = 5$ (left) and for two $MCAN^+$ settings ($\mu = 5$ and $\mu = 10$) for growing dataset (right).

not very significant in a real application scenario, since for $\epsilon = 10$ we have already more than 14 millions of pairs.

In Figure 2 we report the parallel distance computations for increasing values of ϵ as function of dataset size. As explained above, this performance figure can be considered as the parallel cost of the operation. From this experiment, we can see that the algorithm scales well. Note that, the parallel cost of the algorithm remains bounded as the dataset size grows, because, as explained, we correspondingly increment the number peers.

Figure 3 reports the total number of distance computations during the SSJ operation for increasing values of ϵ as function of dataset size. These experiments reflect the intuition that the total number of distance computations is quadratic in the size of dataset. Note also that the total number of distance computations for $MCAN^+$ is much smaller than that for the sequential implementation of the Sliding Window for the same threshold ϵ. For instance, for $\epsilon = 10$, we have about 320 millions of distances for $MCAN^+$ in contrast to more than 21 billions of distances for the sequential Sliding Window (i.e., about 66 times). This means that even if we computed the SSJ sequentially (e.g., as we had all peers in the same physical machine), $MCAN^+$ would take much less time. This should not surprise, since in any case $MCAN^+$ uses a more sophisticated technique to partition the problem (besides the use of the Sliding Window). To give an idea of the computational cost, $MCAN$ with $\mu = 10$ takes about 9.5 sec for completing a SSJ of $\epsilon = 10$.

4. CONCLUSION

Although several research efforts have recently proposed distribute data structures to support similarity range and nearest neighbors queries, there are only few studies that aim at supporting similarity joins. In this article, we have analyzed an implementation strategy for similarity self join based on the $MCAN^+$. This work is inspired by prior work [5] for similarity self joins in metric spaces using a centralized indexing technique called *eD-Index*. We borrowed from the approach of *eD-Index* the idea of overlapping partitions, although our approach is extended to a distributed environment. To the best knowledge of the author, $MCAN^+$ represents the first distributed data structure specifically designed to approach the problem of similarity self join in metric spaces.

The price that we must pay to obtain the results of the SSJ in few seconds instead of waiting hours, is the space occupation. We define the replication factor R of $MCAN^+$ as the ratio N^*/N, where N is the size of the dataset and N^* the number of objects (comprising also replicas) stored in $MCAN^+$. Therefore, it is $R = 1$ for $\mu = 0$ (corresponding with the standard $MCAN$) and it is $R > 1$ for $\mu > 0$.

Experiments of Figure 4 studies how R grows as we increase the number of peers for a fixed dataset size and same $\mu = 5$. It must be highlighted that the extra space due to replication does not grow significantly as we increases the number of peers. In fact, for an $MCAN^+$ with 1,024 peers the replication factor is more or less the double of the one with only two peers. A worst behavior is instead shown by scalability experiments of Figure 4, which shows as R grows when we increase the number of peers to meet increasing sizes of the dataset. It is clear that if we would like to guarantee the scalability of the SSJ response time, we have to tolerate a strong impact in terms of space occupation. However, note that the increment in space is distributed in more peers, for instance, for 1,024 peers we have in average about 4,350 objects for $\mu = 10$ against about 1,000 for $\mu = 0$. The storage cost of this approach is acceptable in a distributed environment such as the one we propose.

An interesting aspect is the high number of results of the similarity self join. As said above, for the most selective query $\epsilon = 0$, we obtain about 2.5 millions of pairs. In a scenario of photo sharing such as the one we propose, it would be helpful to exploit more visual descriptors to reduce the number of matchings. This approach could exploit more $MCAN^+$ overlays, one for each visual descriptor, by intersecting the result sets coming from the execution of the similarity self join on each overlay. The intersection process would not be so expensive, since each object has a unique identifier and therefore it is possible to generate a unique identifier to each pair, which simplifies the intersection process. This is an interesting subject of future work.

5. REFERENCES

[1] Mpeg requirements group, mpeg-7 overview, 2003. Doc. ISO/IEC JTC1/SC29/WG11N5525.

[2] CoPhIR (content-based photo image retrieval), 2008. http://cophir.isti.cnr.it/.

[3] C. Böhm, B. Braunmüller, M. Breunig, and H.-P. Kriegel. High performance clustering based on the similarity join. In *CIKM '00*, pages 298–305, 2000.

[4] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Proximity searching in metric spaces. To appear in *ACM Computing Surveys*, 1999.

[5] V. Dohnal, C. Gennaro, and P. Zezula. Similarity Join in Metric Spaces Using eD-Index. In *Proc. of the14th International DEXA Conference*, volume 2736 of *LNCS*, pages 484–493. Springer, May 2003.

[6] F. Falchi, C. Gennaro, and P. Zezula. A Content-Addressable Network for Similarity Search in Metric Spaces. In *Proc. of the the 2nd International DBISP2P Workshop, Trondheim, Norway*, volume 4125 of *LNCS*, pages 98–110. Springer, August 2005.

[7] F. Falchi, C. Gennaro, and P. Zezula. Nearest neighbor search in metric spaces through content-addressable networks. *Inf. Process. Manage.*, 43(3):665–683, 2007.

[8] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM '01*, pages 161–172, 2001.