Scalable Similarity Self Join in a Metric DHT System[⋆]

Claudio Gennaro

ISTI-CNR
Pisa, Italy
claudio.gennaro@isti.cnr.it

Abstract. Efficient processing of similarity joins is important for a large class of data analysis and data-mining applications. This primitive finds all pairs of records within a predefined distance threshold of each other. 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 bounding 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.

1 Introduction

Similarity join is a database primitive that finds all pairs of objects 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).

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. For illustration, consider a document collection of books and a collection of compact disk documents. A possible search request can require to find *all pairs of books and compact disks which have similar titles.*

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

[⋆] 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.

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* [4, 5], to efficiently support similarity self join searches.

This work refines and extends prior work [6] by simplifying the insertion and the search algorithms, allowing less space occupation, and by providing a new experimental evaluation.

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 [4, 5] a Content-Addressable Network [7] for metric space objects. In the following, we provide a brief overview of the *MCAN*. We then present the *MCAN*$^+$ and show the differences with the original *MCAN* structure.

Our system is based on the well-known CAN (Content-Addressable Network), which is a DHT (distributed hash table) that uses a function for mapping "keys" onto "values" in order to assign them a position in the table. In a CAN, the table is composed of a finite set of individual network nodes (peers). Each peer of the network is dynamically associated with a partition of a n-dimensional Cartesian space. The principle of the CAN is to divide the space in a finite number of distinct rectangular regions, each of them associated to one and only one peer of the network. The peers are responsible for storing and searching of objects covered by their region. Moreover, each peer is aware of the peers that cover adjacent regions, i.e., its neighbors. More precisely, for an n-dimensional space, two regions are neighbors if their sides overlap along $n - 1$ dimensions and are adjacent along one dimension. Since in metric spaces only distance among objects is known and it is not possible to exploit any knowledge of coordinate information, in *MCAN* we use the pivot paradigm for projecting objects of the metric space into n-dimensional vectors.

In particular, let $t_1, \ldots, t_n$ be a set of pivots (usually selected from the metric dataset X), we project an object $x \in \mathcal{D}$, by means of the function F defined as:

$$F(x) : \mathcal{D} \to \mathbb{R}^n \ = (d(x, t_1), d(x, t_2), \ldots d(x, t_n)) \qquad (1)$$

This virtual coordinate space is used to store the object x in the *MCAN* structure, specifically in the peer that owns the zone where the point $F(x)$ 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 d^∞ distance (instead of the Euclidean distance), defined as $d^\infty(x, y) = max_i^n |d(x, t_i) - d(y, t_i)|$. Routing in *MCAN* works in the same manner as for the original CAN structures. An *MCAN* peer maintains a coordinate routing table that holds the IP address and virtual coordinate zones of each of its immediate neighbors in the coordinate space.

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 important 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 $\mathbf{p}.M = \{\mathbf{h}_1, \ldots, \mathbf{h}_h\}$. 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. For further details about *MCAN*, please see [4].

2.1 Similarity Self Join in *MCAN*$^+$

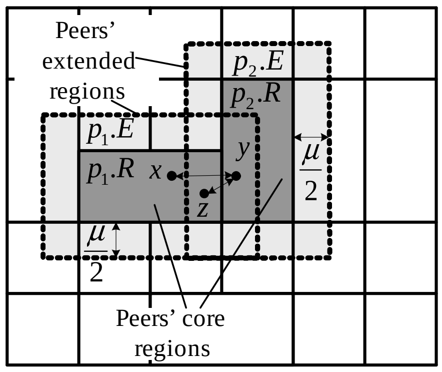
The *similarity join* is a search primitive that combines objects of two sets $X = \{x_1, ..., x_N\}$ and $Y = \{y_1, ..., y_M\}$ into one set such that $X \overset{sim}{\bowtie} Y = \{\langle x_i, y_j \rangle \in X \times Y \mid d(x_i, y_j) \leq \epsilon\}$,. Where the threshold ϵ is a non negative 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 $\mu/2$ peers' bounding regions $\mathbf{p}.R$ equally in all directions so that they overlap their neighbors' regions. This principle ensures that for each pair $\langle x, y \rangle$ for which $d(x, y) \leq \epsilon$, there always exists at least one region of a peer where the pair occurs, for all ϵ such that $0 \leq \epsilon \leq \mu$. Figure 1 illustrates the basic principle of this strategy; the rectangle depicted in grey and bounded by a dashed line represents the region managed by a peer of the *MCAN*$^+$. However, unlike *MCAN* the peer maintains a bounding region that overlaps of $\mu/2$ the regions of its neighbors. The darker innermost rectangles represent the original bounding regions of *MCAN*. 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 above.

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. The SSJ in a peer can be evaluated by a simple exhaustive algorithm that processes all pairs contained in the peer's region. Let m be the number of objects, this algorithm, often referred in literature to as *Nested Loop* (NL), processes about $m^2/2$ pairs. 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 $d(x, y) \leq \epsilon$.

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. However, during the construction of $MCAN^+$ we append extra information with each indexed object. In particular, each metric object x of $MCAN^+$ has the attribute x.Own, which is a set that keeps track of the *id*s of the peers that hold.



$\mathbf{p}_1.id$=1
$\mathbf{p}_2.id$=2
$x.Own$=$\{1\}$
$y.Own$=$\{1, 2\}$
$z.Own$=$\{1, 2\}$
CheckDist(1, $\langle x, y \rangle$)=*true*
CheckDist(1, $\langle z, y \rangle$)=*true*
CheckDist(2, $\langle z, y \rangle$)=*false*

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

Algorithm 1 includes the function *SimilaritySelfJoin*, which takes the *id* k of a peer and the threshold ϵ as input parameters and returns the set of 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 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 *NestedLoop* function, which takes as input the threshold ϵ and returns the set of the pairs that potentially belongs to the result set and which have to be checked using the actual 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 function *CheckDist*, which takes the *id* k of the peer and $\langle x, y \rangle$ as inputs, and returns a boolean that indicates whether or not the pair must be considered. To better understand the behavior of *SimilaritySelfJoin*, please see Figure 1, which illustrates the zones of a peer where an object can lay. The principle of the *CheckDist* algorithm is simple if we observe that the problem of replication occurs when the objects of a pair are owned by distinct peers. The idea here is to exploit the *ids* of the peers (maintained in x.Own) 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 *ids* would work as well).

Algorithm 1 *Similarity Self Join Algortihm*

```
$S := \emptyset$;
for each $\boldsymbol{p} \in$ MCAN$^+$
  $S := S \bigcup$ SimilaritySelfJoin($\boldsymbol{p}$.id, $\epsilon$);
end for each

function SimilaritySelfJoin($k$, $\epsilon$): set
    $P :=$ NestedLoop($\epsilon$);
    $R := \emptyset$;
    for each $\langle x, y \rangle \in P$
        if CheckDist($k$, $\langle x, y \rangle$) then
            if $d(x, y) \leq \epsilon$ then
                $\langle x, y \rangle \rightarrow R$;
            end if
        end if
    end for each
    return $R$;
end function

function CheckDist($k$,$\langle x, y \rangle$):boolean
    CheckDist := false;
    if min($x$.Own $\bigcap$ $y$.Own) $= k$ then
            CheckDist := true;
    end if
end function
```

Algorithm 2 *Insertion Algortihm*

```
procedure Insert($i$, $x$)
    if $\overline{x} \in \boldsymbol{p}(i).R$ then
        $i \rightarrow x$.Own;
        for each $\boldsymbol{h} \in \boldsymbol{p}(i).M$
            $x$.Own := $x$.Own $\bigcup$
                Replicate($\boldsymbol{h}$.id, $x$, $i$);
        end for each
        Store($x$);
    else
        $\boldsymbol{h}$ := GetNrstNghbr($\boldsymbol{p}.M$, $\overline{x}$);
        Insert($\boldsymbol{h}$.id, $x$);
    end if
end procedure

function Replicate($i$, $x$, $j$): set
    if $\overline{x} \in \boldsymbol{p}(i).E$ then
        $i \rightarrow x$.Own;
        for each $\boldsymbol{h} \in \boldsymbol{p}(i).M$
            if $\boldsymbol{h}$.id $\neq j$ then
                $x$.Own := $x$.Own $\bigcup$
                  Replicate($\boldsymbol{h}$.id, $x$, $i$);
            end if
        end for each
        Store($x$);
        return $x$.Own;
    end if
end function
```

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

In order to exploit this SSJ algorithm, *MCAN*$^+$ must employ an insertion algorithm more sophisticated than the one used in *MCAN*. The insertion operation can start from any peer **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 **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 **h** for which $\overline{x} \in \mathbf{h}.R$, 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, which implements the replication principle of *MCAN*$^+$, as described in Algorithm 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). The function returns the set of peers' *ids* that share the object x.

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].

To give an idea of the computational cost of the SSJ, it is useful to observe that the time complexity of the NL algorithm is $\frac{N \cdot (N-1)}{2}$, for which we estimate a computation time of more than five days, using a machine equipped with an Intel 2.13GHz processor. The values of the thresholds ϵ produce a number of pairs that range from about 2.5 millions (for $\epsilon = 0$) to about 4 millions (for $\epsilon = 5$) of the five hundred billions of possible pairs.

We analyzed the behavior of a $MCAN^+$ involving 10 pivots for mapping the metric space in a 10–dimensional vector space. For selection the pivots, we use the Incremental Selection algorithm described in [3]. Since in this work we concentrate our attention on scalability issues, we start from a dataset of size 125,000 and duplicate its size until we reach the maximum size of one million objects. At each step of dataset duplication we try to keep the number of objects in each peer up to about 18,000 (this means about 162 millions of pairs processing which experimentally corresponds to the acceptable computation time of about 80 seconds) by doubling the number of peers accordingly (a peer can be split into two peers maintaining half of the original region). Therefore, datasets of sizes 128,000, 250,000, 500,000, and 1,000,000, were processed using 8, 16, 32, and 64 peers, respectively. The objective of this study is to try to bound the total similarity join computation time by exploiting the parallelism of peer computations. Note that, 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.

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. During the evaluation of the NL in a peer, we can employ the knowledge of the precalculated distances with respect to the reference objects to get a lower bound of the distance $d(x, y)$; allowing us to discard some distance computation. This technique is often known as *pivot filtering*. On the other hand, to give a fair comparison, in the experimental evaluation we consider both the total number of pairs processed by a peer that is given by $m^2/2$ (m is the number of objects of the peer), and the actual number of distance computations (considering the pivot filtering). The former amount can be seen as an upper bound for the computational cost of the SSJ, while the latter one represents in some way a lower bound for the computational cost. The actual computational time will fall in between these two figures, in fact the actual cost may also depend on the cost

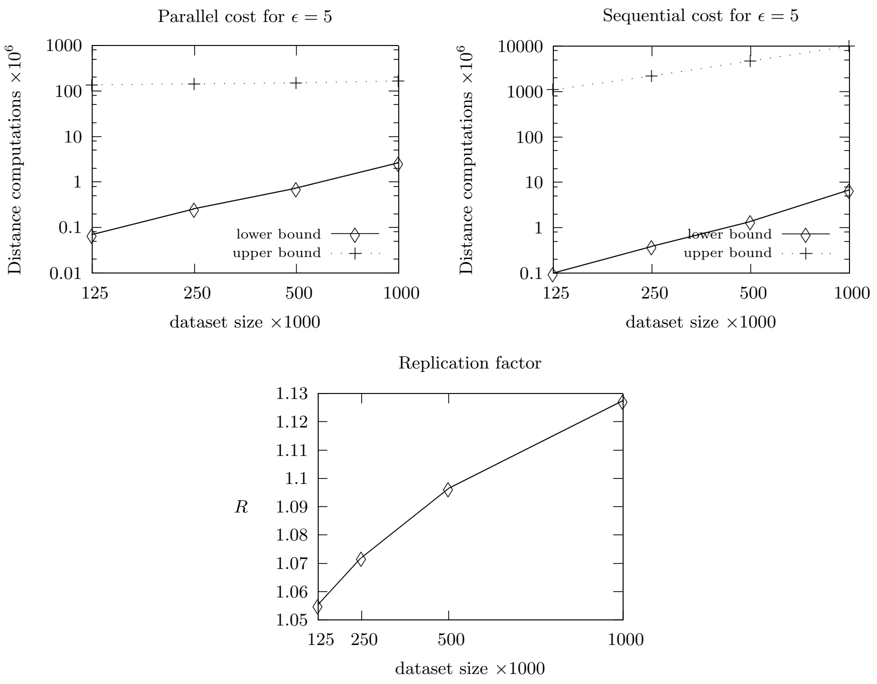


Fig. 2. Parallel and sequential costs, and replication factor of $MCAN^+$ ($\mu = 5$).

for pivot filtering, the cost for implementing the duplicate avoidance, the cost for the disk access, etc.

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

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

In Figure 2 we report the parallel distance computations of an SSJ with $\epsilon = 5$ as function of dataset size.

Figure 2 reports the sequential distance computations during the SSJ operation with $\epsilon = 5$ as function of dataset size. These experiments reflect the intuition that the total number of distance is almost linear in the size of dataset (both axes use logarithmic scales).

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 2 study as R grows when we increase the number of peers to meet increasing sizes of the dataset and same $\mu = 5$. It must be highlighted that the extra space due to replication grows almost linearly as we increases the number of peers.

4 Conclusion

Although a number of distribute data structures have been recently proposed to support similarity range and nearest neighbors queries on metric spaces, there are only few approaches that aim at efficiently supporting similarity joins. In this article, we have analyzed an implementation strategy for similarity self join based on the $MCAN^{+}$ DHT.

We have highlighted the strengths and weaknesses of our solution: if from one hand we are able to approach the problem of quadratic computational complexity of similarity join in terms of search time, on the other hand, we must pay a price in terms of space occupation, which is, however, less than 13% in worst case analyzed.

This problem of space occupation can, however, not only be tolerated in distributed environments such as Peer-to-Peer computing infrastructures, but can be beneficial in terms of fault tolerance, redundancy, and efficiency. Concerning the latter aspect, if the proposed partition scheme is also employed, as we expect, for dealing with range and k-nearest neighbor queries, the replication will have a positive impact to the number of cells involved during the query processing.

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. B. Bustos, G. Navarro, and E. Chvez. Pivot selection techniques for proximity searching in metric spaces. In *Proc. of the XXI Conference of the Chilean Computer Science Society (SCCC'01)*, pages 33–40, 2001.
4. F. Falchi, C. Gennaro, and P. Zezula. A Content Addressable Network for Similarity Search in Metric Spaces. In *Proc. of the 2nd DBISP2P Workshop*, volume 4125 of *LNCS*, pages 98–110. Springer, 2005.
5. 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.
6. C. Gennaro. A Content-Addressable Network for Similarity Join in Metric Spaces. In *Proceedings of the Third International Conference on Scalable Information Systems (Infoscale 2008).* ACM Press, June 2008.
7. S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM '01*, pages 161–172, 2001.