Computer Science and Information Systems 11(2):745–763 DOI: 10.2298/CSIS140215040K

A Distributed Near-Optimal LSH-based Framework for Privacy-Preserving Record Linkage

Dimitrios Karapiperis and Vassilios S. Verykios

School of Science and Technology
Hellenic Open University
{dkarapiperis,verykios}@eap.gr

Abstract. In this paper, we present a framework which relies on the Map/Reduce paradigm in order to distribute computations among underutilized commodity hardware resources uniformly, without imposing an extra overhead on the existing infrastructure. The volume of the distance computations, required for records comparison, is largely reduced by utilizing the so-called Locality-Sensitive Hashing technique, which is optimally tuned in order to avoid highly redundant computations. Experimental results illustrate the effectiveness of our distributed framework in finding the matched record pairs in voluminous data sets.

Keywords: Locality-Sensitive Hashing, Bloom filter, Map/Reduce.

1. Introduction

Recently, a series of bank and insurance company failures triggered a financial crisis of unprecedented severity. Institutions had to engage in negotiations in order to get back on their feet [1]. An important parameter of these negotiations are the customer bases that need to be cleaned up and possibly merged. The process of merging the customer bases and finding out records that refer to the same real entity is known as the Record Linkage, Entity Resolution or Data Matching problem [11]. In our case, we also assume that records belong to different owners, who are bound to protect the privacy of the data they hold by the legislation framework. If privacy should be preserved during the linkage of the records, special techniques should be developed that leverage similarity and simultaneously respect the privacy of sensitive data. This type of linkage is known as Privacy-Preserving Record Linkage and is picking up a lot of steam lately.

More specifically, the process of linking records from various data sets consists basically of two steps. The first step is the *searching* of potentially matching pairs and the second is the actual *matching* of these pairs. The *searching* step refers to the smooth scaling of computational cost in terms of time and consumed resources with respect to the data volume explosion. So long as the data is increasing, we should be able to link records efficiently, in an anticipated manner and offer linkage solutions of low resource consumption. The first step relies on techniques such as the traditional blocking [6], the Mapping-based Indexing [24], the Sorted Neighborhood approach [21], the Q-Gram-based indexing [6] and the Canopy Clustering [15]. The second step, known as the *matching* step, is implemented either in an exact or in an approximate manner. Exact matching of two records can be regarded as a binary decision problem of exact agreement or disagreement. Approximate matching entails the calculation of value similarity, falling in the range of $[0..1]$.

Several methods have been developed for approximate matching of values such as Edit distance [33], Jaro-Winkler distance [46] and SoftTF-IDF [14]. Variations in the data values, mainly due to typographical errors, and privacy concerns, which need to be taken into consideration, lead us to the choice of secure approximate matching [37] solutions for our problem, where value variations are preserved in the transformations that occur to protect their privacy.

By going back to our scenario, let us suppose that two banks started negotiations for a merger and that their corresponding customer bases are voluminous. Since no bank is willing to disclose any confidential data to the other bank, in order to protect their privacy, we should employ high-dimensional data structures to embed that data in, which in combination with the huge collection of records at hand, introduce high computational overhead in the linkage process. Hence, we need an efficient method to identify similar data, given the complex representational structure and the huge size of data sets.

The secure searching solutions, which have been developed to solve the PPRL problem, rely mostly on traditional blocking, where all records that have the same value in a specific field(s) are blocked together for comparison. However, the proposed solutions exhibit a considerable overhead in terms of performance, when applied to voluminous data and especially to high-dimensional data. In [22], as an example, blocking relies on the categorization of records into generalized hierarchies based on the semantics of values of selected fields, which may lead to load imbalance problems, if most values semantically belong to certain categories. Karakasidis and Verykios in [25] present a blocking technique which relies on a sliding window that creates blocks of records. Its performance is considerably degraded, when the size of that window is increased in order to produce more accurate results, as shown experimentally in Sect. 6. Authors in [26], [28] and [17] use redundant probabilistic methods, where each record is blocked into several independent blocking groups, in order to amplify the probability of bringing together similar records for comparison. These techniques though utilize an arbitrary number of blocking groups as well as an arbitrary number of hash functions to shape up the blocking keys for each group. This process has as a result either unnecessary and expensive comparisons or a large number of missed similar record pairs.

In this paper we propose the optimal configuration, denoted by oBfJ, of a naive methodology for PPRL, as introduced in [17]. As shown experimentally, oBfJ can reduce the number of record pairs that are brought together for comparison up to 95% of the total comparison space while it maintains high levels of recall, constantly above 93%. Moreover, by exploiting a number of computational resources of commodity hardware using a distributed framework based on oBfJ, as illustrated in Sect. 5, we will manage these computations with respect to large-scale data volumes. Experimental results in Sect. 6 indicate that our proposed framework outperforms the Multi-Dimensional Privacy-Preserving Blocking (MPPB) method [25], in terms of running time and accuracy in the results.

The structure of the paper is organized as follows. Related work is described in Sect. 2. In Sect. 3 we present an outline of the various building blocks we utilize in our framework. In Sect. 5 we illustrate our proposed framework, which is evaluated in Sect. 6. Conclusions and ideas for future extensions are presented in Sect. 7.

2. Related Work

Several solutions have been presented in the literature in the field of efficient searching for similar records [15, 17, 22–26, 28, 38]. However, these solutions exhibit poor performance when applied to massive data sets. In [15] for example, a cheap distance metric is used for creating clusters of records and then a more expensive, accurate distance metric is used to evaluate the record pairs that are tagged for further evaluation. Nevertheless, the number of record pairs, that should be compared eventually, can still be excessively large. The tree-based indexing methods used in [23, 24, 38] in order to reduce the number of candidate record pairs, as reported and proved in [20, 45] and [4], exhibit quadratic complexity, because they need to scan the whole index structure repeatedly, when these structures are used for representing records even with moderate dimensionality (≥ 10). A detailed survey of blocking techniques for Record Linkage can be found in [12]. An overview of privacy-preserving blocking techniques is provided in [44].

The Bloom filter-based encoding method, as was presented in [40] and is used by our oBfJ methodology, is easily implemented and preserves the similarity of the original records. However under certain circumstances, this method is susceptible to constraint satisfaction cryptanalysis [30]. Scannapieco et al. in [38] present an encoding method that embeds string values in the Euclidean space by using reference sets of random string values. This embedding method imposes certain strict requirements, like the use of random strings of length approximately equal to the length of the values to be embedded, which cannot be applicable to data sets exhibiting quite large variation in the length of their values. Pang et al. in [35] use public reference tables in order to compute the distance of field values from those in the reference tables and to assign them into clusters. However, accurate results are attained only when the reference tables are a superset of the values in the data sets. The encoding method in [13] relies on the extracted q-grams from string values which are sent in an encrypted format to a third-party for comparison. Q-grams are susceptible to frequency attacks and add high communication cost due to the large number of encrypted data that should be sent to the third-party.

Our methodology utilizes a trusted third-party in order to conduct the linkage of the encoded data sets. Two-party techniques, like the ones in [42] and [43], may reduce the risk of privacy breach, like colluding parties, but they are complex and they add high communication cost. Authors in [22, 31], instead of encoding data and submitting it to a third-party, suggest Secure Multi-Party Computation (SMC) protocols [34] to the matching step. These protocols are effective and reliable but they add high computational overhead leading to prohibitive running times.

3. Background and Problem Formulation

For illustration purposes and without loss of generality, let us consider that we have to link two data sets A, which belongs to Alice, and B, which belongs to Bob, as shown in Table 1 and 2 correspondingly. Alice and Bob are allowed to make use of the services offered by an independent party, whom we call Charlie. We assume that Charlie can help in the process by exhibiting what is known as a semi-honest or honest-but-curious behavior [44], which means that even if Charlie is not trying to collude with Alice or Bob, and follows the protocol prespecified by the two custodians, he is still curious to find out as much

information, from what he is presented with, as possible. The cardinality of A and B is N_A and N_B respectively.

Table 1. Data set A belonging to Alice

Id	*FirstName*	*LastName*
1	George	Peters
2	John	Smith

Table 2. Data set B belonging to Bob

Id	*FirstName*	*LastName*
1	William	Grace
2	John	Smyth

Alice and Bob will transform their data sets into secure structures so that linkage will be conducted in a private manner. Also, Alice and Bob need the result of the linkage within a reasonable amount of time, although the cardinality of the corresponding data sets might be millions of records. In the next subsections we present the basic components of a naive methodology for PPRL, as introduced in [17], denoted by BfJ, which relies on the Bloom filter-based encoding method [40] and on the Min-Hash Locality-Sensitive Hashing approach [9]. By applying BfJ and setting certain configuration parameters appropriately, as shown in 4, the truly matched record pairs can be found efficiently and accurately.

3.1. Bloom filters for Private Representation of Data

A string value is anonymized by encoding it as a Bloom filter. A Bloom filter is a data structure used to represent the elements of a set, in order to support membership queries for these elements efficiently, in terms of time and space required [7]. It has been shown in [40] that Bloom filters are able to preserve the distance between the pair of unencoded string values, and for this reason they can be used to compare string values in a private manner. More specifically, a bitmap array of size L, initialized with zeros, is created by hashing all consecutive bigrams of a string (sequences of pairs of adjacent characters), by using F independent composite cryptographic hash functions G_is (see Fig. 1). For example, *MD5* and *SHA1* [39] may play the role of G_is along with other more advanced keyed hash message authentication code (*HMAC*) functions like *HMAC-MD5* and *HMAC-SHA1* [29], which utilize a secret key, that should be shared by the data custodians, for increased security.

Let us suppose that Alice and Bob encode the *FirstName* and the *LastName* attributes of their sets, into field-level Bloom filters with length equal to 6 bits by using one cryptographic hash function for each bigram, as shown in Tables 3 and 4 respectively. By concatenating those field-level Bloom filters for each record, we construct the encoded representations of the original records. The *Id* attribute indicates both the initial *Id* and

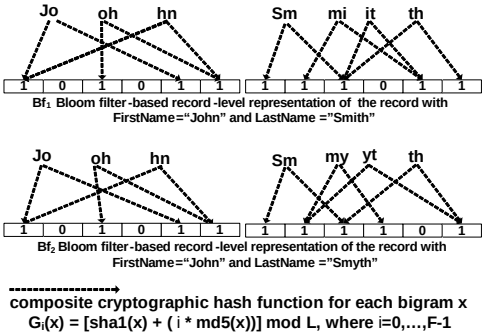

Fig. 1. Concatenation of field-level Bloom filters into an encoded record-level structure

the source data set. Durham in [17, 18] develops a Bloom filter-based record-level encoding format by sampling random bits from field-level Bloom filters either of dynamic or static size she experimentally shows that accuracy is maintained in similarity calculations. The encoded data sets are denoted as A' and B' respectively. The distance of two Bloom filters Bf_1 and Bf_2 can be measured by the Jaccard metric as $d_J(Bf_1, Bf_2) = |BP(Bf_1) \cap BP(Bf_2)| / |BP(Bf_1) \cup BP(Bf_2)|$, where $BP(\cdot)$ returns all bit positions of a Bloom filter set to one.

Table 3. Transforming data set A into A'

Id_A	*Bf*(*FirstName*)	*Bf*(*LastName*)
A1	<1,0,1,1,0,0 >	<1,0,1,1,0,0>
A2	<0,1,1,1,1,0 >	<1,0,1,1,0,0>

Table 4. Transforming data set B into B'

Id_B	*Bf*(*FirstName*)	*Bf*(*LastName*)
B1	<1,1,0,0,1,1 >	<0,0,0,1,0,0 >
B2	<0,1,1,1,1,0 >	<1,0,1,1,0,1>

3.2. Min-Hash Locality-Sensitive Hashing for Efficient Grouping of Similar Bloom filters

An efficient well-known method that is used in finding similar items among huge data sets, is the Locality-Sensitive Hashing (LSH) [19] technique. The LSH technique performs probabilistic dimensionality reduction of high-dimensional data. Locality-sensitive hash functions that are sensitive to the Jaccard metric, comprise the Min-Hash family,

which is denoted by $\mathcal{H}$ [8, 10, 36]. The locality-sensitive property assumes that the probability of generating the same result if we hash two Bloom filters by the same function of $\mathcal{H}$, is dependent upon their Jaccard distance. In essence, the application of the Min-Hash LSH method to the Bloom filters, can be considered as a Λ-independent and redundant blocking technique. In order to generate these Λ independent blocking groups, we should make use of Λ composite hash functions H^j, where $j = 1, \ldots, \Lambda$. Each H^j consists of a fixed number, say K of base hash functions h_k^j, where $k = 1, \ldots, K$ chosen randomly and uniformly from $\mathcal{H}$. The result of each base hash function $h_k^j(\textit{Bf}) = \textit{min}\{\pi_k^j(\textit{Bf})\}$ applied to a Bloom filter *Bf*, is the bit position of the minimum non-zero element after permuting its elements according to the π_k^j, which is the k-th permutation of the j-th blocking group.

For example, say we have to permute $\textit{Bf} = <1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0>$ according to the permutation $\pi_1^1 = <3, 5, 0, 2, 11, 8, 9, 10, 4, 6, 7, 1>$ which transforms *Bf* into $<1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0>$, hence $min\{\pi_1^1(A_1')\} = 3$. Each H^j is used to hashing a Bloom filter to one of the separate blocks of each independent group. By applying to the Bloom filter *Bf* each of the K base hash functions of H^j and by concatenating the results, a $\textit{Key}^j$ is built, which hashes *Bf* to the j-th blocking group, namely $\textit{Key}^j = H^j(\textit{Bf})$, where $H^j = \textit{Concat}(\textit{min}\{\pi_k^j(\textit{Bf})\})$. Each of the blocking groups can be implemented as a simple hash table consisting of key-bucket pairs, where a bucket hosts a linked list where we place the *Id*s of the Bloom filters that have been hashed to this bucket. Another way of implementing blocking groups, especially suited for massive data, is shown in Sects. 5.1 and 5.2.

3.3. Secure Matching of Bloom filter Pairs

Charlie, by using common permutations, hashes the Bloom filters of A and B once to each blocking group. In the blocks of those groups, where Bloom filters of both data sets are located, Charlie compares them in a pairwise manner. To be more precise, the stored *Id*s are used to retrieving the corresponding Bloom filters. A simple decision model classifies those Bloom filter pairs either to matched or to non-matched pairs according to their distance d_J compared to a threshold ϑ, which is defined by the data custodians. Given two Bloom filters $\textit{Bf}_1$ and $\textit{Bf}_2$, we compute the intermediate variables *a*, *b* and *c*, which will be used in the distance calculation of $\textit{Bf}_1$ and $\textit{Bf}_2$, as follows:

$$a = \sum_{l \in [0,L)} [(\textit{Bf}_1[l] == 1) \wedge (\textit{Bf}_2[l] == 1)] \tag{1}$$

$$b = \sum_{l \in [0,L)} [(\textit{Bf}_1[l] == 1) \wedge (\textit{Bf}_2[l] == 0)] \tag{2}$$

$$c = \sum_{l \in [0,L)} [(\textit{Bf}_1[l] == 0) \wedge (\textit{Bf}_2[l] == 1)] \tag{3}$$

The Jaccard distance can be calculated as: $d_J(\textit{Bf}_1, \textit{Bf}_2) = 1 - [a/(a + b + c)]$. We refer the reader to [44] where various classification techniques are discussed. Also, as shown in Fig. 1, all underlying fields of records in A and B, participate equally in the composition of the encoded representation of the record. This means that each field is weighted equally in both blocking mechanism and distance calculation.

3.4. Map/Reduce Framework for Scaling Up Computations

Map/Reduce [16] is a computational paradigm, where an application is divided into fragments of computation, each of which may be executed on any node of a cluster, so that parallelism can speed-up the whole process. The Map/Reduce system runs on top of a distributed file system where data is fragmented into equally-sized subsets and stored on the nodes of a cluster, providing reliability and integrity by managing redundancy of data and scalability by easily adding compute nodes on demand. When a job is submitted to the Map/Reduce framework, map and reduce tasks are allocated to the compute nodes where data, that the job has specified, exists. Each map task, denoted by M_x, processes a subset of data and outputs intermediate <*key, value*> pairs. Each unique *key*, as generated by the map tasks, is distributed by the partitioner tasks to a single reduce task R_x, along with its associated *value*. This mechanism results to the allocation of each unique *key* with its associated vector of *values* to a reduce task which processes this vector of *values* according to the application logic.

4. The oBfJ Methodology

From the LSH theory [19], given two Bloom filters Bf_1 and Bf_2, if it holds that $d_J(Bf_1, Bf_2) \leq \vartheta$ then $\Pr[h_k^j(Bf_1) = h_k^j(Bf_2)] \geq p_\vartheta$, where $p_\vartheta = 1 - \vartheta$ [36] and ϑ is the maximum distance that two Bloom filters should exhibit in order to be considered as similar. Since each H^j consists of K base hash functions, the probability of Bf_1 and Bf_2 colliding in the same block of the j-th group is:

$$\Pr[H^j(Bf_1) = H^j(Bf_2)] \geq p_\vartheta^K. \tag{4}$$

In the naive BfJ (Bloom filters-Jaccard) methodology, the probability of collision is amplified by blocking the Bloom filters independently to Λ blocking groups. While, though, the probability of collision is amplified, the redundant blocking groups increase the running time and the utilized space. Therefore, the naive BfJ should be optimized in order to produce the smallest set of Bloom filter pairs that include as many as possible from the matched pairs and simultaneously to consume the least possible computational resources. For a given value of K, by setting $\Lambda_{opt} = \lceil \ln(\delta) / \ln(1 - p_\vartheta^K) \rceil$ [5, 19], each Bloom filter pair is returned by this scheme with probability at least $(1 - \delta)$, where δ is an input parameter indicating the probability of failing to return a matched pair. For example, for a distance threshold $\vartheta = 0.40$, by setting $K = 5$ and $\delta = 0.1$, we generate $\Lambda_{opt} = 29$ blocking groups. By utilizing this structure, theoretical guarantees are provided that each matched Bloom filter pair is returned with probability at least 0.9. An arbitrary value for Λ ($\Lambda > 28$), would return the matched Bloom filter pairs but would result to useless additional running time and space. Since the Bloom filter-based method is highly accurate and distance-preserving, by finding the matched Bloom filter pairs, we also find the original matched record pairs, which are a subset of those Bloom filter pairs.

The choice of the optimal value for K, that is K_{opt}, is more complex since we should choose a value that minimizes the running time, which actually depends on the number of distance computations. We estimate the expected running time $E[RT]$ by using several different values of K, and their corresponding Λ_{opt} values, and we choose the one that minimizes the $E[RT]$ value. In the same way as authors do in [41], we decompose $E[RT]$

into two basic components: $E[RT] = RT_T + E[UC]$. The first component RT_T, which is equal to

$$RT_T = \Lambda_{opt}\, K(N_{A'} + N_{B'}), \tag{5}$$

is the time required to compute the Key^js for each record of A' and B'. The second component $E[UC]$ is the expected number of unique collisions (unique pairs) in the blocks. Let $Q(\tau)$ denote the probability of a collision for a Bloom filter pair, exhibiting distance τ. Then, $Q(\tau)$ is equal to $1-(1-(1-\tau)^K)^{\Lambda_{opt}}$ [5]. A tight estimation of $E[UC]$ is given by $E[UC] = \sum_{\iota=1}^{N_{A'}} \sum_{\kappa=1}^{N_{B'}} Q(d_J(Bf_\iota, Bf_\kappa))$. However, it is not practical to compute $E[UC]$ in this way since it includes the calculation of distances of the whole comparison space $A' \times B'$. By exploiting statistical information about the distribution of the Bloom filter pairs in distance intervals and by using the Monte Carlo method [32], we can make inferences about the number of Bloom filter pairs falling to those intervals. More specifically, we sample the minimum required number of pairs for each interval from the comparison space and by computing their distances, we make global inferences about the Bloom filter pairs distribution to those distance intervals. For example, for our data sets, the probability that a Bloom filter pair falls within the distance interval $[0.80, 0.85)$ is 0.23. This probability distribution follows the Poisson distribution with $\lambda = 16$ (the 16th interval $[0.80, 0.85)$ starting from $[0.0, 0.05)$). Given such empirical data, let $\widehat{p}_\alpha$ be the estimated proportion of pairs that fall to the α-th interval. Let also $\widehat{N}_\alpha$ be the estimated number of those pairs which equals to $\widehat{p}_\alpha\ |A' \times B'|$. Then, the estimated number of unique collisions, denoted by $\widehat{UC}$, for a given K and Λ_{opt} is

$$\widehat{UC} = \sum_{\alpha=1}^{N_{int}} \widehat{N}_\alpha\, Q(\widetilde{X}_\alpha), \tag{6}$$

where N_{int} is the number of the intervals and $\widetilde{X}_\alpha$ is the midpoint of each interval. Thus, by setting several values for K, we compute for each of them, Λ_{opt} and $E[RT]$. The value of K that exhibits the smallest $E[RT]$, is chosen as K_{opt}. For example, by setting $K = 2, \ldots, 12$, we observe from the curve in Fig. 2 that as K approaches 5 from the left, $E[RT]$ decreases, it is minimized when $K = 5$ and for $K > 5$, $E[RT]$ grows substantially. Therefore, the value of 5 is chosen as K_{opt}.

We can also set K empirically since the correctness of the scheme is guaranteed by setting appropriately Λ_{opt} , but by doing so we may not be optimal in terms of running time.

5. A Distributed LSH-based Framework

In this section we present a distributed framework that relies on the Min-Hash LSH technique for searching similar records efficiently and on the Map/Reduce programming paradigm in order to enable scalability, by providing compute nodes on the fly. Alice and Bob submit their data formatted as shown in Tables 3 and 4 to the trusted third party, Charlie, who utilizes a Map/Reduce system on top of a distributed file system to run the computations. Input data sets are horizontally partitioned into independent subsets D^s of equal predefined size, and then each subset is replicated ρ times on different compute

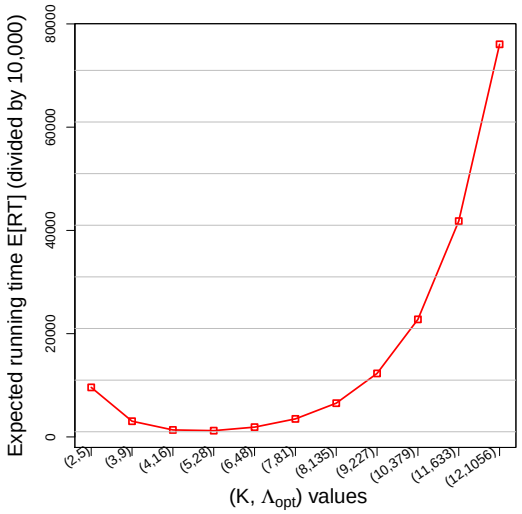

Fig. 2. Expected running time $E[RT]$ for specific (K, Λ_{opt}) values

nodes, for fault tolerance purposes. Map tasks receive the subset D^s stored on the node they run. To illustrate the framework, Charlie's cluster consists of two nodes C_1 and C_2 that hold data and a master node C_0 which coordinates the functionality of the cluster. The master node splits each data set as submitted by Alice and Bob in two subsets of equal size. Each subset in our example has one row. By setting the replication factor ρ to 1, each subset of A' and B' is distributed to different nodes. For example, Bf_{A2}, Bf_{B1} are stored on C_1 and Bf_{A1} and Bf_{B2} are stored on C_2. If we set $\rho = 2$, then each subset should be replicated twice, which would result to the total omnipresence of subsets to nodes. We present two large-scale linkage strategies that use the described cluster of nodes. The first strategy, denoted by St_1, consists of one Map/Reduce job, while the second strategy, that is St_2, utilizes two chain Map/Reduce jobs. Both strategies have two distinct logical steps, (a) the generation of the Key^js and (b) the matching of the formulated Bloom filter pairs.

5.1. Generation of the Key^js

When Charlie submits the Map/Reduce job of linkage, Algorithm 1 runs on each node by the map tasks M_1 and M_2. For each Bloom filter contained in a subset D^s, the Key^js are constructed by applying the corresponding permutations, as agreed by Alice and Bob and shown in Algorithm 1 in line 7. For illustration purposes, by letting $K = 2$, Λ_{opt} is set to 2 (to be precise Λ_{opt} should be set to 6, given $\delta = 0.1$ and $\vartheta = 0.40$). On the compute node C_1, which holds Bf_{A2} and Bf_{B1}, M_1 builds the following Key^js for Bf_{A2}: <1B,3,1> and <2B,0,5>. The map task M_2, running on C_2, builds also for Bf_{B2} the same Key^js. The labels 1B and 2B denote that the corresponding Key^js refer to the first and the second blocking group respectively. The Key^js, along with the corresponding Ids,

are emitted to the partitioner tasks (Algorithm 1, line 10). This phase is common for both linkage strategies.

Algorithm 1 Transformation of Bloom filters into $<Key^j, Id>$ pairs in the map phases of both St_1 and St_2

Input: D^s, π_k^js permutations, K
Output: list ($<Key^j, Id>$)

```
 1: Λ_opt = computeOptimal(K)
 2: for i=1 to |D^s| do
 3:    Id ← D_i^s.Id
 4:    Bf ← D_i^s.Bf
 5:    for j=1 to Λ_opt do
 6:       for k=1 to K do
 7:          Key^j = Concat(min{π_k^j(Bf)})
 8:       end for
 9:    end for
10:    emit(Key^j, Id)
11: end for
```

5.2. Matching of the Formulated Bloom Filter Pairs

The *Id*s of the Bloom filters corresponding to a unique Key^j are routed to the same reduce task R_x, by the partitioner tasks. Then in strategy St_1, each R_x calculates the Jaccard distance of each Bloom filter pair, as illustrated in Sect. 3.3 and shown in Algorithm 2, lines 2 and 4. If the distance of a pair is below or equal to a predefined threshold (ϑ) then it is classified as a matched pair, otherwise as a non-matched pair (Algorithm 2 lines 7, 9). In our running example, the Key^1=<1B,3,1>, which refers to the first blocking group and it is generated by both Bf_{A2} and Bf_{B2}, has as a result their grouping to the same block and their classification as a matched pair. It is also noted in Fig. 3, where the strategy St_1 is outlined, that these Bloom filters also exhibit the same keys in the second blocking group, which results to a redundant distance computation.

The notation $[Id]$ in Algorithm 2, in the input statement, denotes that we get a list of *Id*s for each Key^j. In lines 3 and 5 function $get(\cdot)$ retrieves a Bloom filter from a data store. The same function uses a naive cache mechanism in order to cache locally a limited amount of the retrieved Bloom filters, for efficient use in subsequent computations that these Bloom filters participate. In our implementation the data store is a relational database. Linking two data sets of $200,000$ Bloom filters each, the Min-Hash scheme produces around $130,000,000$ pairs ($K = 5, \Lambda_{opt} = 29$), including duplicates which may be up to 12% of the total number of pairs.

In order to avoid the calculation of distance of duplicate pairs across the R_xs, in strategy St_2, we employ two chain Map/Reduce jobs. The map phase of the first job includes the generation of the Key^js, as illustrated in Algorithm 1, while during the reduce phase, the Bloom filter pairs are formulated without computing their distance (Algorithm 3).

Algorithm 2 Matching of the grouped Bloom filters in the reduce phase in St_1

Input: list(<Key^j, [Id] >)
Output: list ({"M","U"}, Id_A, Id_B)
1: **for each** Key^j **do**
2: **for each** $Id_A \in A'$ **do**
3: $Bf_A \leftarrow get(Id_A)$
4: **for each** $Id_B \in B'$ **do**
5: $Bf_B \leftarrow get(Id_B)$
6: **if** $d_J(Bf_A, Bf_B) \leq \vartheta$ **then**
7: output("M", Id_A, Id_B);
8: **else**
9: output("U", Id_A, Id_B);
10: **end if**
11: **end for**
12: **end for**
13: **end for**

Algorithm 3 Formulation of Bloom filter pairs in the first reduce phase in St_2

Input: list(<Key^j, [Id] >)
Output: list (Id_A, Id_B, ·)
1: **for each** Key^j **do**
2: **for each** $Id_A \in A'$ **do**
3: **for each** $Id_B \in B'$ **do**
4: output(Id_A, Id_B, ·);
5: **end for**
6: **end for**
7: **end for**

The second map phase is an identity function that simply passes its input data to the second reduce phase, where unique record pairs, as represented by their *Id*s, are distributed to the reduce tasks which perform the distance computations. This generating-unique-pairs strategy limits the comparisons only to the unique Bloom filter pairs, as specified by oBfJ. However, the use of a relational database to provide the Bloom filters results to long running time, because each pair requires the retrieval of two Bloom filters since the probability of having at least one of them cached is very low. On the contrary to St_1, as illustrated in line 3 of Algorithm 2, the Bloom filter of A' is retrieved once from the database, it is then cached and compared to all the Bloom filters of B' for the specific Key^j (line 5). The two strategies are outlined in Figs. 3 and 4 respectively.

6. Evaluation

In the experiments we evaluate oBfJ in terms of (a) the accuracy in finding the encoded matched record pairs, (b) the accuracy in finding the truly matched record pairs, (c) the efficiency in reducing the number of candidate record pairs and (d) the scalability in voluminous data sets. For (a),(b) and (c) we use semi-synthetic data sets of 50,000 records each, extracted from the NCVR [2] list. Each record includes 4 fields, namely *Id*, *LastName*,

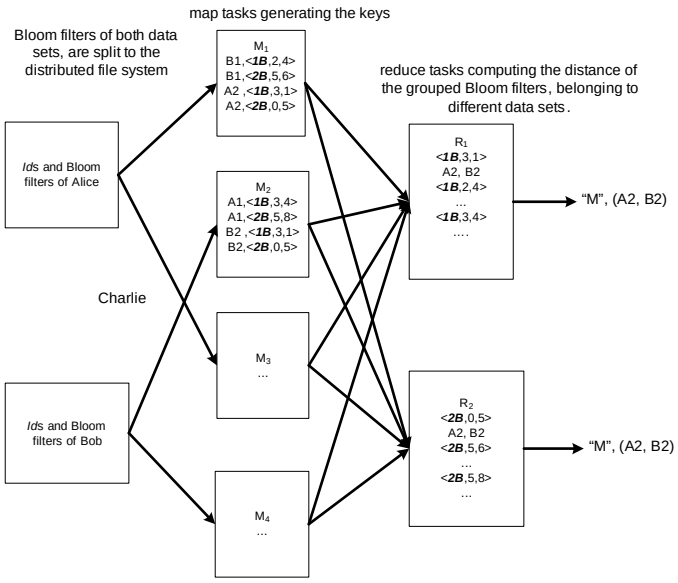

Fig. 3. Outline of strategy St_1

Algorithm 4 Matching of the grouped Bloom filters in the second reduce phase in St_2

Input: list(Id_A, Id_B, [·])
Output: list ({"M","U"}, Id_A, Id_B)
1: $Bf_A \leftarrow get(Id_A)$
2: $Bf_B \leftarrow get(Id_B)$
3: **if** $d_J(Bf_A, Bf_B) \leq \vartheta$ **then**
4: output("M", Id_A, Id_B);
5: **else**
6: output("U", Id_A, Id_B);
7: **end if**

FirstName and *Address*. We develop a software prototype that extracts data sets, A and B of user-defined size and perturbation frequency, from the NCVR list [2]. Records are chosen randomly for perturbation, according to the specified frequency. For each perturbed record, two fields are chosen randomly and undergo (a) a single perturbation scheme (Pt_1) or (b) a double perturbation scheme (Pt_2). Insert, edit, delete and transpose operations are used to perturb the values of those chosen fields. The perturbation frequency of the records is set to 0.3. Field-level Bloom filters are created with size L equal to 500 bits, by using 15 cryptographic hash functions for each bigram, as proposed in [40]. All experiments, except for the large-scale ones (see Sect. 6.3), are executed in a dual-core Pentium PC of 8 GB RAM.

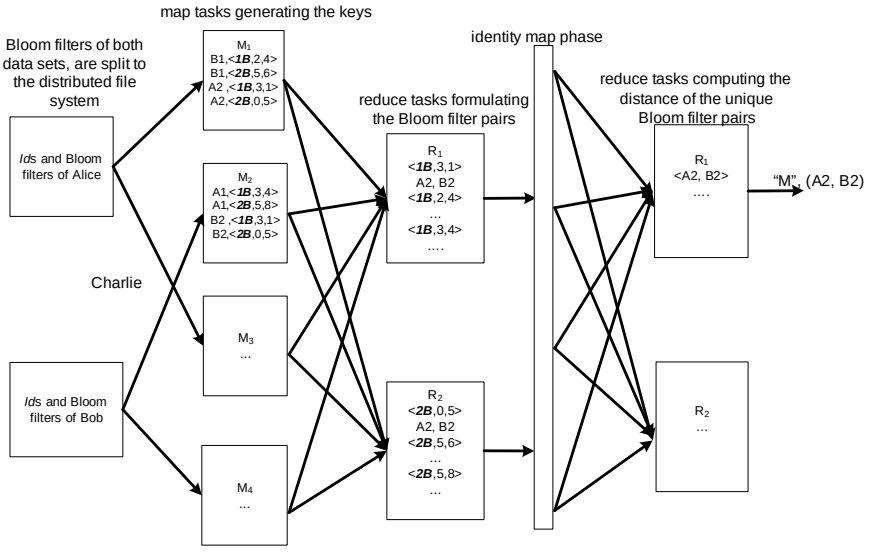

Fig. 4. Outline of strategy St_2

6.1. Selected Measures

The Pairs Completeness (*PC*), the Pairs Quality (*PQ*) and the Reduction Ratio (*RR*) metrics [12] are employed to evaluate the efficiency of our methodology in finding the truly matched record pairs with respect to accuracy. The set of the identified encoded matched record pairs is denoted by $\mathcal{M}$ and the set of the original matched record pairs by M. Accuracy is measured by the PC_M metric, which is equal to $|\mathcal{M} \cap M|/|M|$. The efficiency in generating mostly matched pairs with respect to the number of the candidate pairs is indicated by the *PQ* metric which is equal to $|\mathcal{M} \cap M|/|CR|$, where *CR* is the set of the candidate record pairs. The Reduction Ratio illustrates the percentage in the reduction of the comparison space and it is equal to $1.0 - |CR|/|A' \times B'|$.

6.2. Comparative Results

The efficiency of the Min-Hash LSH scheme and the accuracy of the Bloom filter-based encoding jointly determine the overall performance of our methodology. In Fig. 5 we demonstrate the results by performing experiments for various values for K ($\vartheta = 0.40$ and $\delta = 0.1$) along with their corresponding Λ_{opt} values, namely $(5, 29)$,$(6, 49)$,$(7, 82)$ and $(8, 136)$. Furthermore, for a specific value of K, we perform experiments by setting Λ to several arbitrary values. For example, for $K = 5$, we set $\Lambda = <49, 82, 136>$ or for $K = 8$, we set $\Lambda = <29, 49, 82>$. We observe from Fig. 5, for a given value of K, as Λ approaches from the left to Λ_{opt}, the PC_M rate increases but after Λ_{opt}, the PC_M rate remains almost stable. This implies that the largest part of the matched Bloom filter pairs is identified (each pair is returned with probability at least 0.9) and consequently the same happens for the largest part of the original matched record pairs. An amount of

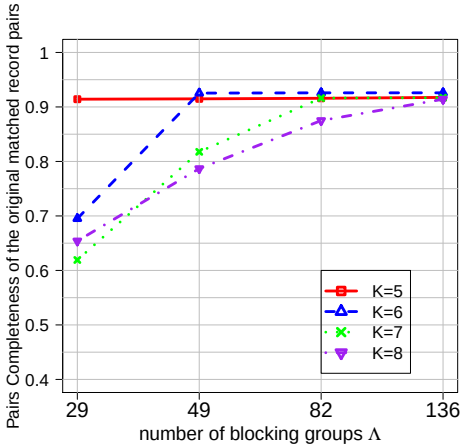

Fig. 5. Trying several values of K along with the corresponding Λ_{opt} values

the original matched record pairs is missed, approximately 8%, due to improper Bloom filter-based encoding, where the initial distances are not preserved.

We compare our methodology to the Multidimensional Privacy-Preserving Blocking (MPPB) method as proposed in [25]. MPPB is based on the K-Medoids algorithm [27] for creating clusters from elements of public reference sets. Next, records are classified to those clusters and they are encoded into Bloom filters, where the Sorted Neighborhood algorithm [21] is used for the selection of the Bloom filter pairs for comparison within each cluster. The Dice coefficient [11] is employed to measure the similarity between those pairs. Thresholds in the comparisons are set to 0.85 and 0.80 for the two perturbation schemes respectively.

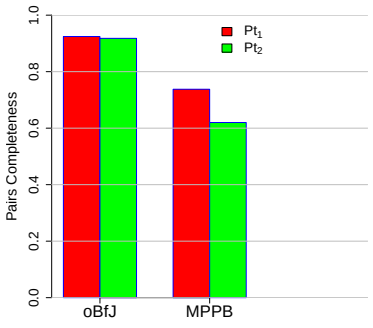

Fig. 6. The Pairs Completeness rates of the original matched record pairs

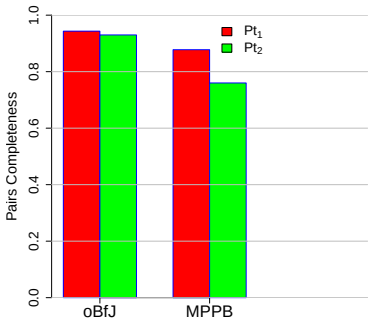

Fig. 7. The Pairs Completeness rates of the matched Bloom filter pairs

We set $K_{opt} = 5$ and $\delta = 0.1$ as the basic parameters of oBfJ which yield $\Lambda_{opt} = 29$ blocking groups for the Pt_1 scheme with $\vartheta = 0.40$ and $\Lambda_{opt} = 45$ groups for the Pt_2 scheme with $\vartheta = 0.45$. Duplicate distance computations are prevented by utilizing an efficient $\mathcal{O}(1)$ structure (like a HashMap in Java programming language) which stores the *Id*s of each compared pair only once, therefore discarding any computation that has already been conducted. Highly accurate results are generated by oBfJ, as shown in Fig. 6, with PC_M rate above 92%. In order to increase the PC_M rate for the MPPB method, we had to increase the window size to a large value (> 70), which had dramatic impact on the performance and a slight increase in the PC_M rate, clearly outperformed by oBfJ, as shown in Figs. 6 and 7.

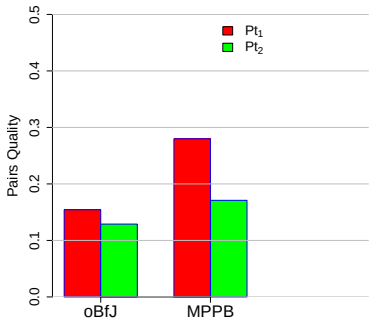

Fig. 8. The Pairs Quality *PQ* rates

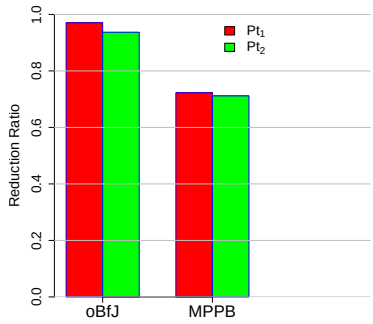

Fig. 9. The Reduction Ratio *RR*

Although, oBfJ displays high *RR*, constantly above 94% (Fig. 9), the *PQ* rates remain rather low (Fig. 8). Therefore, although oBfJ is efficient in reducing the initial $|A' \times B'|$ number of pairs, there are still candidate non-matched Bloom filter pairs with distance d_J close to ϑ but slightly above it. The MPPB method displays better *PQ* rate but lower *RR*.

6.3. Scalability

Scalability is measured in terms of execution time, utilizing a Map/Reduce system. More specifically, Apache Hadoop 1.0.4 is used, as the Map/Reduce implementation, running on a cluster of four compute nodes, which are virtual machines of *Okeanos* [3], the cloud service of the Greek Research and Academic Community. Two data sets are linked, which consist of $300,000$ records. By adding more physical compute nodes in the reduce phase, execution time is decreased as shown in Fig. 10. Permutations are shared across the map tasks by utilizing the distributed file system, since each task runs on its own execution environment. It is clearly shown that by utilizing a relational database, strategy St_2 becomes inefficient and its main feature, which is the unique distance computation of duplicate pairs, does not contribute a lot to the overall system's performance.

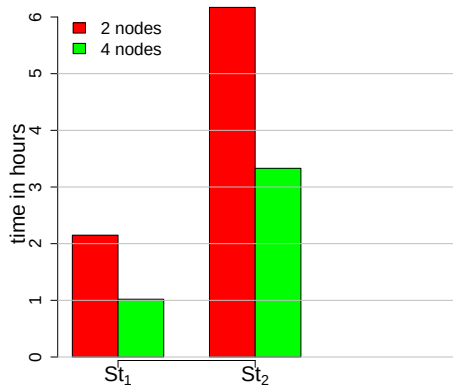

Fig. 10. Execution time exhibited, utilizing a Map/Reduce system using two concrete strategies

7. Conclusions

Linking huge collections of anonymized records is an intriguing problem in the core of the domain of Privacy-Preserving Record Linkage. In this paper, we introduce an LSH-based distributed method on top of a Map/Reduce computational paradigm. Records are encoded into Bloom filters, in order to protect privacy of the underlying data, and then they are submitted to a trusted third party, which splits and distributes them to a file system with replication capabilities. The huge collection of records demands the utilization of a large number of compute nodes that leverage scalability. The Min-Hash LSH technique is applied by producing from each Bloom filter some Key^js which correspond to some blocking groups. The number of those blocking groups (Λ_{opt}) and the number of the hash functions for each Key^j (K_{opt}) are optimized with respect to accuracy and performance. Bloom filters are retrieved from a relational database and they are cached locally to each node for subsequent computations. Those Bloom filters that exhibit identical Key^js are grouped together and their Jaccard distance is calculated on a pairwise manner. We believe that the utilization of a more sophisticated and robust cache mechanism in combination with the use of a distributed storage system, that facilitates efficient queries to massive data sets, is an interesting research direction that may boost the performance of strategy St_2.

References

1. Financial Times US bank mergers (2010), http://www.ft.com/cms/s/0/5e969dc0-c01f-11df-b77d-00144feab49a.html
2. North Carolina voter registration database (2013), ftp://www.app.sboe.state.nc.us/data

3. Okeanos, GRNET's cloud service for the Greek research and academic community (2013), https://okeanos.grnet.gr/home/
4. Aggarwal, C., Yu, P.: The igrid index: Reversing the dimensionality curse for similarity indexing in high dimensional space. In: Proc. 6th ACM international conference on Knowledge discovery and data mining (SIGKDD). pp. 119 – 129 (2000)
5. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Communications of the ACM 51(1), 117 – 122 (2008)
6. Baxter, R., Christen, P., Churches, T.: A comparison of fast blocking methods for record linkage. In: Proc. ACM Workshop on Data Cleaning Record Linkage and Object Consolidation (SIGKDD). pp. 25 – 27 (2003)
7. Bloom, B.: Space/time trade-offs in hash coding with allowable errors. Communications of ACM 13(7), 422 – 426 (1970)
8. Broder, A.Z.: On the resemblance and containment of documents. In: Proc. Compression and Complexity of Sequences. pp. 21 – 29 (1997)
9. Broder, A.Z., Charikar, M., Frieze, A., Mitzenmacher, M.: Minwise independent permutations. In: Proc. 30th ACM symposium on Theory of computing (STOC). pp. 327–336 (1998)
10. Broder, A.Z., Glassman, S., Manasse, M., Zweig, G.: Syntactic clustering of the web. In: Selected papers from the 6th international conference on World Wide Web archive. pp. 1157 – 1166 (1997)
11. Christen, P.: Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Springer, Data-Centric Systems and Applications (2012)
12. Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. IEEE Transactions on Knowledge and Data Engineering (TKDE) 12(9) (2012)
13. Churches, T., Christen, P.: Some methods for blindfolded record linkage. BioMed Central Medical Informatics and Decision Making 4(9) (2004)
14. Cohen, W., Ravikumar, P., Fienberg, S.: A comparison of string distance metrics for name-matching tasks. pp. 73 – 78 (2003)
15. Cohen, W., Richman, J.: Learning to match and cluster large high-dimensional datasets for data integration. In: Proc. 8th ACM international conference on Knowledge discovery and data mining (SIGKDD). pp. 475 – 480 (2002)
16. Dean, J., Ghemawat, S.: Mapreduce: Simplifed data processing on large clusters. Communications of the ACM 51(1), 107 – 113 (2008)
17. Durham, E.: A Framework For Accurate Efficient Private Record Linkage. Ph.D. thesis, Vanderbilt University, US (2012)
18. Durham, E., Kantarcioglu, M., Xue, Y., Toth, C., Kuzu, M., Malin, B.: Composite Bloom filters for secure record linkage. IEEE Transactions on Knowledge and Data Engineering 99(PrePrints) (2013)
19. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proc. 25th International Conference on Very Large Databases (VLDB). pp. 518–529 (1999)
20. Goodman, J., O'Rourke, J., Indyk, P.: Handbook of Discrete and Computational Geometry. CRC (2004)
21. Hernandez, M., Stolfo, S.: Real world data is dirty: Data cleansing and the merge/purge problem. Data Mining And Knowledge Discovery 2(1), 9 – 37 (1988)
22. Inan, A., Kantarcioglou, M., Bertino, E., Scannapieco, M.: A hybrid approach to private record linkage. pp. 496 – 505. Proc. IEEE 24th International Conference on Data Engineering (ICDE) (2008)
23. Inan, A., Kantarcioglu, M., Ghinita, G., Bertino, E.: Private record matching using differential privacy. In: Proc. 13th International Conference on Extending Database Technology (EDBT). pp. 123 – 134 (2010)
24. Jin, L., Li, C., Mehrotra, S.: Efficient record linkage in large datasets. pp. 137 – 146. 'Proc. 8th International Conference on Database Systems for Advanced Applications (DASFAA) (2003)

25. Karakasidis, A., Verykios, V.: A sorted neighborhood approach to multidimensional privacy preserving blocking. In: IEEE International Conference on Data Mining Workshops. pp. 937 – 944 (2012)
26. Karapiperis, D., Verykios, V.: A distributed framework for scaling up LSH-based computations in privacy preserving record linkage. In: Proc. 6th Balkan Conference in Informatics (BCI). pp. 102 – 109. ACM (2013)
27. Kaufman, L., Rousseeuw, P.: Clustering by means of medoids. Reports of the Faculty of Mathematics and Informatics (1987)
28. Kim, H., Lee, D.: Fast iterative hashed record linkage for large-scale data collections. In: Proc. 13th International Conference on Extending Database Technology (EDBT). pp. 525 – 536 (2010)
29. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: keyed-hashing for message authentication, internet rfc 2104 (1997), http://tools.ietf.org/html/rfc2104
30. Kuzu, M., Kantarcioglou, M., Durham, E., Malin, B.: A constraint satisfaction cryptanalysis of Bloom filters in private record linkage. In: Proc. 11th international conference on Privacy enhancing technologies (PETS). pp. 226 – 245 (2011)
31. Kuzu, M., Kantarcioglu, M., Inan, A., Bertino, E., Durham, E., Malin, B.: Efficient privacy-aware record integration. In: Proc. 16th International Conference on Extending Database Technology (EDBT). pp. 167 – 178 (2013)
32. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
33. Navarro, G.: A guided tour to approximate string matching. ACM Computing Surveys 33(1), 31 – 88 (2001)
34. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Eurocrypt. pp. 223 – 238 (1999)
35. Pang, C., Gu, L., Hansen, D., Maeder, A.: Privacy-preserving fuzzy matching using a public reference table. Intelligent Patient Management 189, 71 – 89 (2009)
36. Rajaraman, A., Ullman, J.: Mining of Massive Datasets, chap. Finding Similar Items. Cambridge University Press (2010)
37. S., T.: Privacy–preserving string comparisons in record linkage systems: a review. Information Security Journal: A Global Perspective 17(5), 253 – 266 (2008)
38. Scannapieco, M., Figotin, I., Bertino, E., Elmagarmid, A.: Privacy preserving schema and data matching. In: Proc. ACM international conference on Management of data (SIGMOD). pp. 653 – 664 (2007)
39. Schneier, B.: Applied cryptography: protocols, algorithms, and source code in C. Wiley (1996)
40. Schnell, R., Bachteler, T., Reiher, J.: Privacy-preserving record linkage using Bloom filters. BMC Medical Informatics and Decision Making 9 (2009)
41. Shakhnarovich, G., Darrell, T., Indyk, P.: Locality-sensitive hashing using stable distributions. In: Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing). MIT Press (2004)
42. Vatsalan, D., Christen, P., Verykios, V.: An efficient two-party protocol for approximate matching in private record linkage. pp. 125 – 136 (2011)
43. Vatsalan, D., Christen, P., Verykios, V.: Efficient two-party private blocking based on sorted nearest neighborhood clustering. pp. 1949 – 1958 (2013)
44. Vatsalan, D., Christen, P., Verykios, V.: A taxonomy of privacy-preserving record linkage techniques. Information Systems (Elsevier) 38(6), 946 – 969 (2013)
45. Weber, R., Schek, H., Blott, S.: A quantitative analysis and performance study for similarity search methods in high dimensional spaces. In: Proc. 24th International Conference on Very Large Data Bases (VLDB). pp. 194 – 205 (1998)
46. Winkler, W.: String comparator metrics and enhanced decision rules in the Fellegi–Sunter model of record linkage. American Statistical Association pp. 778 – 783 (1990)

Dimitrios Karapiperis is a PhD student with the Hellenic Open University. He also holds an MSc degree from the University of York (UK) and a BSc degree from the Technological Institute of Thessaloniki (Greece.). His research interests lie primarily in the area of Privacy-Preserving Record Linkage where he focuses on developing similarity algorithms, data structures, approximation schemes and scalable solutions that rely on various randomization schemes. More specifically, he deals with Locality-Sensitive Hashing-based efficient searching techniques, secure matching on Bloom filter-based encoded data and distributed/scalable solutions using the Map/Reduce programming paradigm.

Vassilios S. Verykios received the Diploma degree in Computer Engineering from the University of Patras in Greece in 1992, and the MS and the PhD degrees from Purdue University in 1997 and 1999, respectively. From 1999 to 2001 he was with the Faculty of Information Systems in the College of Information Science and Technology at Drexel University, Pennsylvania, as a tenure track Assistant Professor. From 2001 to 2005 he held various research positions at Intracom SA, SingularLogic SA and CTI and visiting Assistant Professor positions in the University of Thessaly, Hellenic Open University and the University of Patras. From 2005 to 2011 he was an Assistant Professor in the Department of Computer and Communication Engineering at the University of Thessaly in Volos, Greece. Since January of 2011, he is an Associate Professor in the School of Science and Technology, at the Hellenic Open University in Patras, Greece.

Received: February 15, 2014; Accepted: June 10, 2014.