MapReduce Implementations for Privacy Preserving Record Linkage

Dimitris Boussis
Computer Engineering and Informatics Department
University of Patras, Greece
bousis@ceid.upatras.gr

Elias Dritsas
Computer Engineering and Informatics Department
University of Patras, Greece
eldritsas@gmail.com

Andreas Kanavos
Computer Engineering and Informatics Department &
Hellenic Open University
kanavos@ceid.upatras.gr

Spyros Sioutas
Department of Informatics
Ionian University, Corfu, Greece
sioutas@ionio.gr

Giannis Tzimas
Computer & Informatics Engineering Department
TEI of Western Greece
tzimas@teimes.gr

Vassilios S. Verykios
Hellenic Open University,
Patras, Greece
verykios@eap.gr

ABSTRACT

Over the last decade, the vast explosion of Internet data has fueled the development of Big Data management systems and technologies. The huge amount of data in combination with the need for records linkage under privacy perspective, has led us to current study. To this direction, we describe Privacy Preserving Record Linkage problem based on Bloom Filter encoding techniques which both maintain users' security and permit similarity control. Moreover, we extended our study to the $HLSH/FPS$ private indexing technique and briefly describe four implementations in the MapReduce distributed environment that is capable of processing large scale data. We also conducted experimental evaluation of these four versions in order to evaluate them in terms of job execution time, memory and disk usage [1].

Keywords

Privacy Preserving Record Linkage; Hadoop; MapReduce; Bloom Filters

1. INTRODUCTION

The rapid evolution of technology and Internet has created huge volume of data at very high rate, deriving from commercial transactions, social networks, scientific research. The mining and analysis of this volume of data may be beneficial for the humans in crucial areas such as health, economy, national security, leading to more qualitative results.

A common problem in data analysis is the record linkage process (RL) which finds records in a dataset that refer to the same entity across different data sources (e.g., data files, books, websites, and databases) [1],[3],[7],[11].

The purpose of RL is to categorize all possible combinations of records from different databases as similar or dissimilar by using attributes that are not necessarily identifying fields. The RL model requires at least two members that will provide their data in form of tables. A table row corresponds to an entity that is described by the columns. Often, the model of the RL is simplified by having only two members that provide the data to be combined (Alice & Bob) with or without the presence of a third member (Carol). The third member undertakes the interconnection process and communicates process results to participant members.

Privacy-preserving policies often prevent research into personal data. Thus, organizations are legally and ethically bounded to exchange sensitive personal data, leading to datasets that are either free of sensitive personal data or encrypted to greatly enhance privacy protection. The privacy requirement during the RL process paved the way to Privacy-Preserving Record Linkage ($PPRL$) [4],[12],[13]. As in the case of RL, the $PPRL$ process find pairs of records referred to the same entity from multiple data sources where the classification, as similar or dissimilar, is conducted based on encoded data to avoid disclosure of confidential data about the entities presented in the problem.

An efficient blocking scheme for $PPRL$ is the $HLSH/FPS$ when combined with Bloom-Filter based encoding. It utilizes Locality Sensitive Hashing and frequent collision tables to Hamming distances between Bloom Filter based encoded pairs of records in order to reduce the number of pairs when a more rigid similarity comparison is performed. $PPRL$ blocking techniques fall into the batch processing category and in the Big-Data world one of the most used system for batch processing applications is MapReduce which is distributed and fault-tolerant. In this paper, we evaluate the performance of four MapReduce work-flows of the LSH/FPS blocking scheme for the $PPRL$ framework.

The rest of this paper is organized as follows: Section2 analyzes background knowledge around the encoding techniques based on Bloom Filter and private indexing. Also, Section 3 briefly describes the MapReduce framework of the $HLSH/FPS$ implementations. Finally, Section 4 presents experimental evaluation and Section 5 presents conclusions.

[1]This work was supported in part by Hellenic Foundation for Public Scholarships and in part by University of Patras.

SETN '18, July 9–15, 2018, Rio Patras, Greece

DOI: https://doi.org/10.1145/3200947.3201043

2. RELATED WORK

2.1 PPRL encoding techniques

In this section we describe some Bloom filter based encoding techniques that are necessary for the $PPRL$ process.

2.1.1 String encoding

The basic idea of this approach is the hashing of q-grams in string fields of records based on Bloom Filters [10]. Bloom Filters [2] consist of a set of K hash functions. Their result puts the position of a bit in a bit vector of size S. Their objective is to give us a quick answer for the membership of an element to a set, by controlling K-positions in the bit vector. The computation of K hash functions $H_i(x)$ may be done through two independent hash functions as:

$$H_i(x) = (h_1(x) + i \cdot h_2(x)) \mod S \quad (1)$$

For the $h_1(x)$ and $h_2(x)$ functions, we choose the cryptographic methods $HMAC - SHA1$ and $HMAC - MD5$ respectively due to their widespread and efficient implementations on cryptographic platforms.

2.1.2 Record encoding

This method is used for records encoding instead of string, as previous method is used for. Each record consists of fields such as name, username, age, address, etc. As the $PPRL$ process aims at the protection of such data, it is necessary to encode the values of the selected fields from all table's records. To this direction, we suggest an encoding method based on a Bloom Filter for which it is necessary to pre-select values for the involved elements and Bloom Filters, such as the average number q-grams. Three different approaches for records encoding using Bloom Filter are described below.

FBF (Field-level-Bloom-Filters) **encoding** [5],[6],[10] is considered as the simplest extension of string data encoding with Bloom Filter. The field values of a record are encoded on separate Bloom Filters, which then compose a larger Bloom Filter that will be used for encoding the entire record. In brief, the encoding steps are:

1. For a selected Q value (the number of q grams), we calculate the average number of q-grams g of each record for fields that will participate in the $PPRL$ process and calculate Bloom filter's appropriate size S_{FBF}.
2. From each string field, q-grams are extracted for a selected Q value.
3. Exported q-grams are encoded by S_{FBF} size Bloom Filters using Fragmentation Components.
4. Bloom Filters produced from each field are combined into a larger one with predetermined series of joins.

The FBF encoding is distinguished in FBF/Static and FBF/dynamic. The first one requires the definition of Q, K and S_{FBF} to encode the records, while the second one, for given Q values, imposes an initial preconditioning step to calculate the average number of q-grams g so as to calculate the appropriate Bloom Filter size S_{FBF}.

The basic idea of **CLK** (Cryptographically Long Keys) **encoding** [11] is the use of large size S Bloom Filter to encode all fields of the record by using the produced q-grams of each field for a selected Q value and K hash functions. The encoding steps are:

1. Export q-grams from each field for a selected Q value.
2. Union of all produced sets of q-grams of the values to be encoded.
3. Extracted q-grams are placed on a S-size Bloom Filter using K hash functions.

Since CLK encoding places common q-grams from different fields to the same K locations in the Bloom Filter, it is difficult for the attacker to perceive either the encoding parameters or the original field values. Also, this particularity of CLK encoding, i.e. common q-grams between fields, can lead to incorrect results in the similarity control. As an example, regarding names "James Johnson" and "John Jameson", while being dissimilar, similarity control over CLK encodings may decide that these entries are similar.

The **RBF** (Record-level-Bloom-Filter) **encoding** is based on FBF and attempts to enhance privacy protection in the $PPRL$ process introducing additional parameters and information into the encoding steps [5],[6]. Initially, it encodes the values of the fields based on separate Bloom Filters and in following creates a random set of bits from each one so as to compose a larger Bloom Filter. Finally, it applies a random bit permutation of the larger Bloom Filter with RBF encoding being the result of this rearrangement.

With regard to the number of bits required to be selected for the encoding of each field, we consider two ways of calculating it, uniform and weighted. The first way uses uniform bit selection from the FBF encoding of the record while the second one uses weighted. Uniform selection of bits requires equal or approximately equal number of bits for each field, i.e. S_f. Weighted way uses a weighted selection of the field encoding bits which leads to selecting more or fewer bits for some of the fields. More to the point, in [5],[6], it is mentioned that the weighted choice is based on the importance of each field in the interconnection process. In order to discover the significance of the fields in the process, the probabilities m and u of the Fellegi-Sunter probability model are used. The weights of agreement and disagreement as well as the range of these two weights are calculated and the normalized percentage of the range of each field is then calculated. In this way, each field contributes a percentage of w_i to the final Bloom Filter. The size of the final Bloom Filter S_{RBF} is derived from the w_i percentage that maximizes that size.

2.2 Private Indexing

The goal of indexing in $PPRL$ is to substantially reduce the pairs of encoded records to be tested through similarity control. In this case, the third member (Carol) has little information about data encoding of Alice and Bob. To this direction, we are going to discuss $HLSH$ indexing.

The **HLSH** (Hamming Locality Sensitive Hashing) **Indexing** [5],[8] is used for partitioning private records encoded in binary form of length S. Let T_l be a set of $l = 1, \ldots, L$ independent hash tables consisting of dynamic sets of key-values. Each hash table T_l uses a set of K hash functions h_l^k that return the value of a randomly selected bit from the binary hashed rows in the table. The values of K functions are a key to the encoded records and can gather from Alice's and Bob's encoded sets A' and B'. Entries from two sets are stacked in the same key for a T_l hash table thus recommending a possibly identical pair if they match in K bits. Id fields' values of encoded records can be in following used in order to finally form possibly identical pairs.

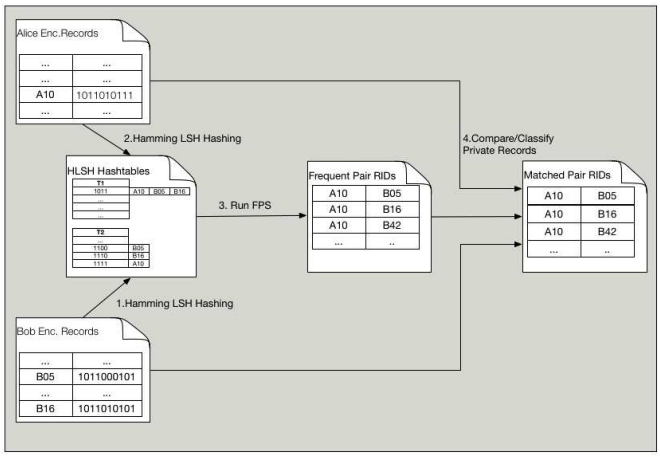


Figure 1: HLSH under FPS

Let encoded records $r_A \in A'$ and $r_B \in B'$ consist of an Id and the fields Bf_A and Bf_B respectively. In addition, let i be a selected value as a limit for the Hamming metric calculated from the equation $dH = |Bf_A \otimes Bf_B|$. We consider a family H of hash functions having the following property:

$$if \;\; d_H \leq \theta \;\; then \;\; Pr[h_k^l(Bf_A) = h_k^l(Bf_B)] \leq p_\theta \qquad (2)$$

$$k = 1, 2, \ldots, K \;\; l = 1, 2, \ldots, L \;\; p_\theta = 1 - \frac{\theta}{S} \qquad (3)$$

The suitable value for the number of hash functions K can be empirically computed, as the accuracy of the method is mainly based on the number of tables, L_{opt}. Generally, this value should form enough buckets so that the number of interconnected lists for the pairs of records is low; for bigger values, more identical entries appear in pairs of records. The formation of a pair of identifiers or encoded entries $\{r_A, r_B\}$ during $HLSH$ in one of the T_l tables, is called collision. The method is redundant so a pair can occur at $C = 1, \ldots, L_{opt}$ hash tables. The pair $\{r_A, r_B\}$ with collisions $C = L_{opt}$ is with high probability similar and intuitively one can argue that as the number of conflicts increases, then it is more likely for the records to be the same.

3. MAPREDUCE FRAMEWORK

$HLSH$ along with use of Frequent Pair Schema (FPS) [9] can lead to fast and efficient record sharing by checking similarity of frequent collision pairs. We present four MapReduce implementations of the $HLSH/FPS$ method for different size of encoded records of Alice A', Bob B', hash tables Tl as well as set of candidate IDs R_{Ids}. We consider that set B' is smaller than set A', so that it is chosen for the initial creation of the hash table T_l (Figure 1).

The use of $HLSH/FPS$ allows the implementation of an effective system with a relative low memory footprint. Our investigation focuses on memory saving and suggest four different versions of the $HLSH$ methodology, namely $v0$, $v1$, $v2$ and $v3$. Each version is based on assumptions for almost all sizes of the problem and progressively "transfers" these sizes from the slow disk to the faster memory of the Mapper/Reducer. We assume that every Mapper or Reducer in a MapReduce task has a fixed memory limit m_{task} that can be committed by YARN. Each of the four versions consists of 2 or 3 different MapReduce Jobs, which in following consist of a number of tasks (Mappers or Reducers) depending on the HDFS size of the problem and user settings.

Version $v0$ is characterized by memory saving when performing the job, but is very expensive in disk use and is especially suitable for Apache YARN environments with low memory availability for the tasks of a MapReduce job. It assumes that Alice's and Bob's encoded records and T_l are so large that cannot be available in the limited task memory. All pairs of identifiers from the $HLSH$ process are formed and in following, the ones that appear at least C_f times are stored again in HDFS to be loaded into memory of the last job that undertakes the interconnection of the proposed IDs based on the identifiers. This approach, in addition to multiple MapReduce tasks, can be considered as the most naive as it forms all pairs of identifiers that can be derived from the $HLSH$ process. On the contrary, it is the version that uses the least memory in Mapper/Reducer tasks according to experimental results.

Version $v1$ allows more relaxed conditions for the committed memory of the tasks to be performed. We assume that the set of T_l tables is able to fit into memory of each task in a MapReduce job. Having this important information in mind, we can perform the $HLSH/FPS$ by exclusively storing the often conflicting pairs in HDFS.

In the last two versions, we also assume that the records of the smaller set B' are capable of being stored as a whole in the m_task memory of Mappers and Reducers. In both versions, the first job resembles the creation and storage in HDFS of the hash table T_l of the set of records. In the second work, the two versions are differentiated in terms of use or non-use of the Reduce phase.

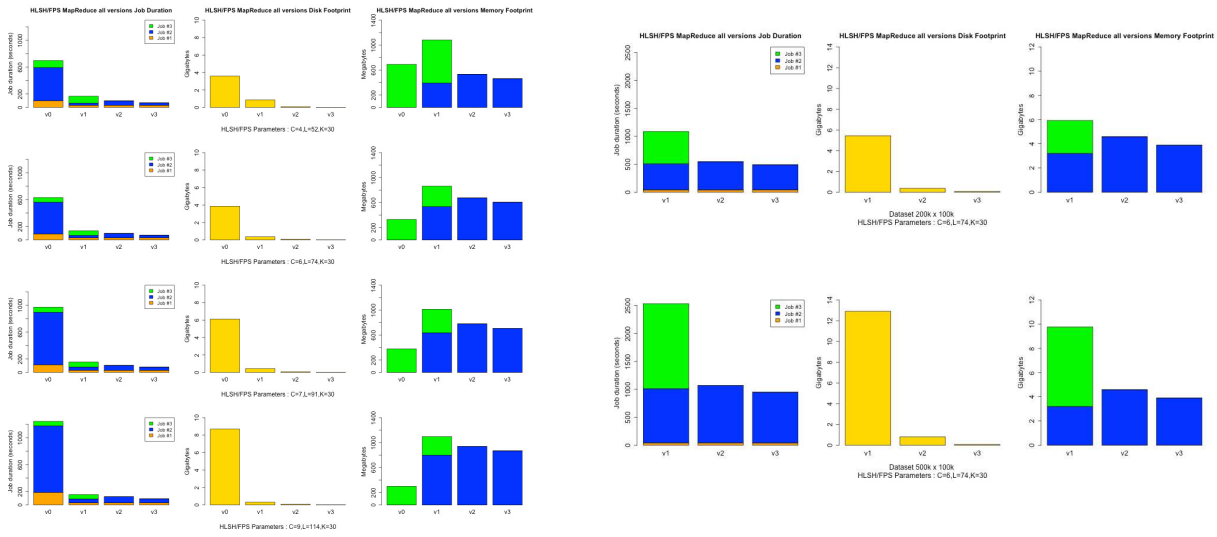
4. PERFORMANCE EVALUATION

The evaluation of four schemas is conducted by considering CLK encoding for $PPRL$ process for $S = 4096$ under the following settings for the parameters δ[2], C_f, L_{C_f}, K.

δ	C_f	L_{C_f}	K
0.001	4	52	30
0.0001	6	74	30
0.00001	7	91	30
0.000001	9	114	30

In the first screen of Figure 2, the simulation results of four versions of $HLSH/FPS$ are shown. The fastest in all cases is $v3$, while $v0$ has the largest footprint on disk since it writes to HDFS all pairs of identifiers derived from $HLSH$. We also observe that for the highest value of δ, the footprint of jobs total memory is also large, but as δ decreases and simultaneously $HLSH$ parameters change, then memory consumption decreases as the number of candidate records for comparison increases. Regarding other versions, as the philosophy of FPS strategy is utilzed, execution times are slightly affected by the change. The disk footprint for $v1$ is slightly affected; the same stands for $v2$ and $v3$. However, it is evident that as L_{C_f} increases, while the number

[2]δ is the confidence parameter defining the likelihood that pairs which are actually the same, are not matched in the tables. This value is usually low, indicatively $\delta = 0.01$.



(a) (b)

Figure 2: PPRL evaluation

of candidate records to be compared is reduced, the memory footprint for all operations, except for $v0$, grows faster. As the number of records of B' remains constant, this increase corresponds to the increase of the hash table T_l.

We then conduct the same procedure for the two largest sets of records, without the $v0$ version. In this case, we show measurements for $\delta = 0.0001$ as well as $C_f = 6$, $L_{C_f} = 74$ and $K = 30$. The second screen of Figure 2 presents the prevalence of versions $v3$ and $v2$ on $v1$ on all metrics.

5. CONCLUSIONS

The four versions that are presented give Carol member the capability to choose between the slow, but memory economical, and the fastest, but demanding, task MapReduce executions. It also shows the need for implementing techniques that help users to decide (in terms of resource use, relative costs and problem size) which of the four versions is appropriate. The experimental evaluation shows that versions $v1$, $v2$, $v3$, which make progressively smarter memory usage, have the advantage of quick execution of $HLSH/FPS$ compared to $v0$. But as the number of records grows, the demanded size to be put into its memory also increases. With the prospect of slow but integrated $HLSH/FPS$ process, $v0$ may be the best proposal for Hadoop environments with limited memory resources.

6. REFERENCES

[1] R. Baxter, P. Christen, T. Churches, et al. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD*, volume 3, pages 25–27, 2003.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] P. Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media, 2012.

[4] C. Clifton, M. Kantarcioglu, A. Doan, G. Schadow, J. Vaidya, A. K. Elmagarmid, and D. Suciu. Privacy-preserving data integration and sharing. In *SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*, pages 19–26, 2004.

[5] E. A. Durham. *A framework for accurate, efficient private record linkage*. PhD thesis, University of Texas at Dallas, 2012.

[6] E. A. Durham, M. Kantarcioglu, Y. Xue, C. Toth, M. Kuzu, and B. Malin. Composite bloom filters for secure record linkage. *IEEE transactions on knowledge and data engineering*, 26(12):2956–2968, 2014.

[7] I. P. Fellegi. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[8] D. Karapiperis and V. S. Verykios. An lsh-based blocking approach with a homomorphic matching technique for privacy-preserving record linkage. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):909–921, 2015.

[9] D. Karapiperis and V. S. Verykios. A fast and efficient hamming lsh-based scheme for accurate linkage. *Knowledge and Information Systems*, 49(3):861–884, 2016.

[10] R. Schnell, T. Bachteler, and J. Reiher. Privacy-preserving record linkage using bloom filters. *BMC medical informatics and decision making*, 9(1):41, 2009.

[11] R. Schnell, T. Bachteler, and J. Reiher. A novel error-tolerant anonymous linking code. *German Record Linkage Center*, 2011.

[12] S. Trepetin. Privacy-preserving string comparisons in record linkage systems: a review. *Information Security Journal: A Global Perspective*, 17(5-6):253–266, 2008.

[13] D. Vatsalan, P. Christen, and V. S. Verykios. A taxonomy of privacy-preserving record linkage techniques. *Information Systems*, 38(6):946–969, 2013.