Integrity for Approximate Joins on Untrusted Computational Servers

Sabrina De Capitani di Vimercati[1], Sara Foresti[1], Sushil Jajodia[2], Stefano Paraboschi[3], Pierangela Samarati[1]

[1] Università degli Studi di Milano – 26013 Crema, Italy *firstname.lastname*@unimi.it
[2] George Mason University – Fairfax, VA 22030-4444 jajodia@gmu.edu
[3] Università di Bergamo – 24044 Dalmine, Italy parabosc@unibg.it

Abstract. In the last few years, many efforts have been devoted to the development of solutions aiming at ensuring the confidentiality and integrity of data and computations in the cloud. In particular, a recent solution for verifying the integrity of equi-join queries is based on the insertion of checks (markers and twins) whose presence provides probabilistic guarantees on the integrity of the computation. In this paper, we propose an approach for verifying the integrity of *approximate join queries*, which is based on the introduction of a discretized version of the join attribute and on the translation of the approximate join into an equi-join defined over the discrete attribute added to the original relations. The proposed approach guarantees the correctness and completeness of the join result, while causing a limited overhead for the user.

1 Introduction

Cloud computing has brought enormous benefits in terms of the availability of a universal access to data as well as of elastic storage and computation services. More and more often users and organizations put their (possibly sensitive) data in the hands of external cloud providers, which become responsible for the storage and management of such data [5,10,16]. A recent trend in cloud computing is a distinction between providers of *storage* services and providers of *computational* services. This diversification supports the development of efficient applications that combine the functions offered by different cloud providers. In this context, users and organizations can therefore decide to store their data at reliable and well-known storage servers and perform computationally intensive processes (e.g., join operations) using the computational services offered by a less expensive and potentially untrusted computational server. Besides performance considerations, an important advantage of relying on storage and computational servers is due to the economic advantage of such a choice [3]. While appealing, this approach brings inherent risks related to the confidentiality and integrity of data and computations, which are difficult to mitigate since data are not under the direct control of their owners. The research community has dedicated many efforts in developing solutions for these problems, resulting in several approaches

to protect the confidentiality and integrity of data at rest (e.g., [5]), as well as of computations over them (e.g., [12,17,18]).

In this paper, we make a step forward and present a solution for verifying the integrity of *approximate join* queries. An approximate join aims at combining tuples with similar (even if not equal) values for the join attribute, and can be needed in several applications (e.g., to detect duplicate entities in different databases or to identify data clusters). The current techniques can verify the integrity of equi-join queries only (e.g., [3]) and then cannot be directly applied to verify the integrity of approximate joins. Moreover, since data are typically encrypted to protect their confidentiality, the evaluation of similarity conditions characterizing an approximate join cannot be efficiently executed on such encrypted data. A client is then not able to delegate the join operation to a computational server without revealing the plaintext values of the join attribute. In the remainder of this paper, after the presentation of some basic concepts and of the problem we aim at addressing (Section 2), we illustrate an approach for verifying the integrity of approximate joins (Section 3). Our solution consists in adding to the original relations a discretized version of the join attribute, translating approximate joins into equi-joins over the discretized attribute. The equi-join is computed as a semi-join, delegating to an external computational server the execution of the join, which is a computationally intensive operation. The techniques in [3] are used to verify the integrity of the computation performed by the computational server (Section 4). Our solution does not impact the correctness and completeness of the join result, and provides limited overhead for the storage servers and for the user (Section 5).

2 Basic concepts and problem statement

We consider a scenario where a client wishes to evaluate an *approximate join* between two relations B_l and B_r stored at two trustworthy storage servers $\mathcal{S}_l$ and $\mathcal{S}_r$, respectively. The computation of the approximate join is delegated to an external and potentially unreliable computational server $\mathcal{C}_s$. Intuitively, an approximate join between B_l and B_r matches tuples that are sufficiently similar, meaning that the values of their join attribute are similar. The similarity between the values of the join attribute can be measured by choosing a distance function (the Euclidean distance in our scenario) and a threshold α set by the client. The query formulated by the client is of the form "SELECT A FROM B_l JOIN B_r ON $|B_l.I - B_r.I|<\alpha$ WHERE C_l AND C_r AND C_{lr}," with A a subset of attributes in $B_l \cup B_r$; I the set of join attributes; $|B_l.I - B_r.I|<\alpha$ the similarity condition and α the threshold fixed by the client; and C_l, C_r, and C_{lr} Boolean formulas of conditions over attributes in B_l, B_r, and $B_l \cup B_r$, respectively. The evaluation of conditions C_l and C_r is pushed down to the storage servers.

Current approaches for integrity verification consider only equi-joins that are executed as semi-joins (or regular joins) by a computational server and are based on the combined adoption of *encryption on the fly* (to protect data confidentiality), and of *markers* and *twins* (to provide integrity guarantees) [2,3].

Each storage server first receives from the client the sub-query it should evaluate and the information necessary for the adoption of encryption on the fly, markers, and twins. It then executes the received sub-query (obtaining relations L and R) and projects the join attribute (obtaining relations LI and RI), thus naturally removing duplicate values. Each storage server then duplicates the tuples in its relation that satisfy a twinning condition C_{twin} defined by the client on the join attribute (to guarantee that twins belong to the join result). Twinned tuples are made unrecognizable to the computational server by combining the value of the join attribute with a random salt before encryption. Each storage server also inserts fake tuples (markers), not recognizable as such by the computational server, into the relation before sending it to the computational server. Markers generated by the two storage servers have the same values for the join attribute (to guarantee their presence in the join result), and these values do not appear in real tuples (to avoid spurious tuples). The resulting relations LI^* and $RI^*$are encrypted by the storage servers (obtaining relations LI^*_k and RI^*_k), with a key communicated by the client and that changes at each query, and are sent to the computational server. The computational server evaluates the equi-join between the two relations received from the storage servers and sends the result JI^*_k to the client. The client decrypts JI^*_k, verifies its integrity (i.e., the client checks whether all expected markers are in the result and twinned tuples do not appear solo), and removes markers and twins (obtaining relation JI). The client then sends JI to both the storage servers, which return to the client all the tuples in L and R having a value for the join attribute in JI. Upon receiving these relations (LJ and RJ) the client recombines them with JI obtaining the join result. Figure 1 illustrates an example of execution of an equi-join, assuming to adopt one marker (with value m for the join attribute), and to twin tuples whose join attribute is equal to a or d.

The semi-join approach mentioned above cannot be used for approximate joins. In fact, the computational server should evaluate a similarity condition $|L.I - R.I|<\alpha$ over encrypted attributes, which is possible only if the encryption function supports the evaluation of arithmetic operations. Such encryption functions (e.g., homomorphic encryption) are inefficient and not suitable for all scenarios. Also, the definition of markers and twins should be revised to comply with the similarity condition, without revealing the nature of the tuples in the encrypted relations.

3 Approximate join transformation

Our approach for verifying the integrity of an approximate join is based on a discretization process applied on the domain of the join attribute. This discretization allows us to translate an approximate join into an equi-join. For simplicity, we assume that the domain D of the join attribute is the set of real, natural, or integer numbers. We note however that our solution can be extended to operate also over any domain characterized by a total order relationship.

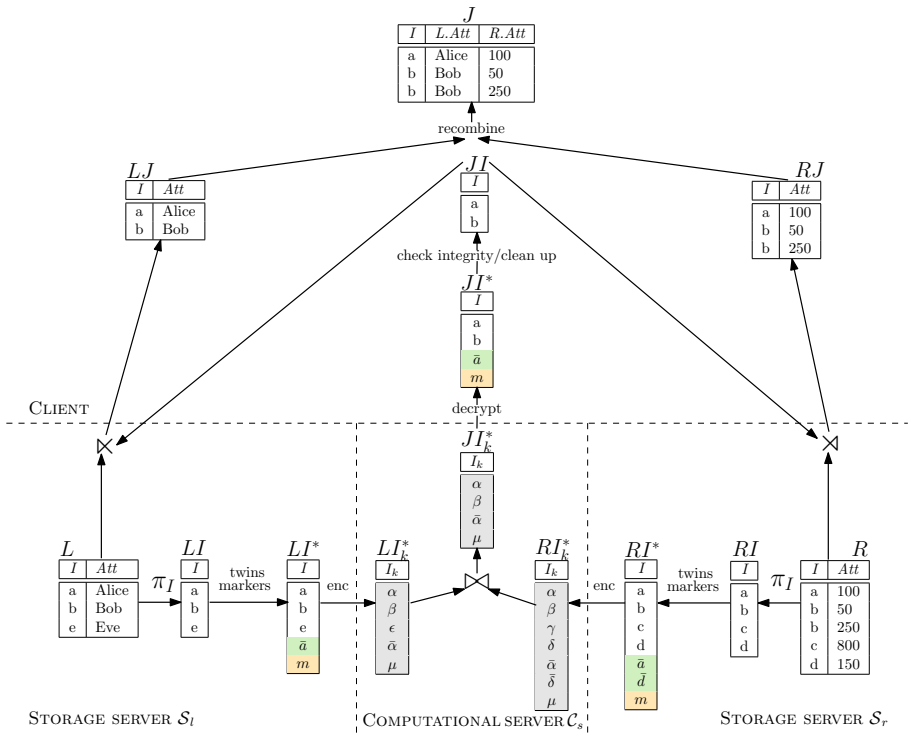


Fig. 1. Equi-join execution as a semi-join

3.1 Discretized domain

The discretization process redefines the domain of the join attribute, transforming it into a (coarser grained) discrete domain. In this way, the similarity condition $|L.I - R.I| < \alpha$ can be transformed into an equality condition over the discretized join attribute, transforming the approximate join into an equi-join.

The discretization of domain D requires to define a *discrete domain* $\hat{D}$ and a *mapping function* f that maps original values into discrete values. To this aim, we chose a *granularity* γ of the discrete domain, which corresponds to the distance between two consecutive values in $\hat{D}$, and a *reference point* p. The reference point p is a value in the original domain D that belongs also to the discretized domain $\hat{D}$ and that, together with γ, can be used to determine the values in $\hat{D}$. We define the values in $\hat{D}$ to be at a distance multiple of γ from p. Formally, a discretized domain is defined as follows.

Definition 1 (Discretized domain). *Let D be a continuous domain, γ be a granularity, and p be an element in D. A discretized domain $\hat{D}$ of D is defined as the set of values in D whose distance from p is a multiple of γ, that is, $\hat{D} = \{v \in D : v - p = x\gamma$, with $x \in \mathbb{Z}\}$.*

CUSTOMER – L

Name	Availability
Alice	11
Bob	48
Carol	18
David	62

PRODUCT – R

Id	Price
p_1	27
p_2	8
p_3	14
p_4	46

(a) original relations

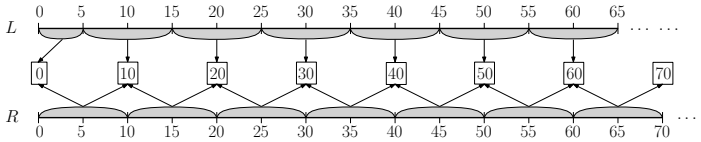
$\hat{L}$

Name	Availability	$\hat{I}$
Alice	11	10
Bob	48	50
Carol	18	20
David	62	60

$\hat{R}$

Id	Price	$\hat{I}$
p_1	27	20
p_1	27	30
p_2	8	0
p_2	8	10
p_3	14	10
p_3	14	20
p_4	46	40
p_4	46	50

(b) discretized relations



(c) mapping

Fig. 2. An example of two relations (a), their discretization with $\alpha = 5$ (b), and the corresponding mapping of the join attribute (c)

For instance, a discretized domain of the natural numbers $\mathbb{N}$, which is the domain of attribute `Availability` in relation CUSTOMER in Figure 2(a), is domain $\hat{D}$=$\{0, 10, 20, \ldots\}$, assuming 0 as reference point and 10 as granularity.

The discretized domain of the join attribute should be the same for both the relations involved in the join operation to permit the correct evaluation of the equi-join condition between them. However, the mapping function used to map each tuple in L to a value in $\hat{D}$ (i.e., the partitioning of D into intervals of size γ and their association with discrete values) may be different from the mapping function used for the tuples in R, as discussed in the following.

3.2 Choosing the correct granularity

Each relation stored at the two storage servers $\mathcal{S}_l$ and $\mathcal{S}_r$ is complemented with an additional attribute, denoted $\hat{I}$, whose values have been obtained through the discretization of the corresponding join attribute domain. The similarity condition $|L.I - R.I|<\alpha$ is then transformed into an equi-join condition of the form $L.\hat{I} = R.\hat{I}$. The set of tuples returned by the evaluation of this equi-join condition should be *correct*, meaning that all tuples satisfying the original similarity condition should be part of the result, and should include a *limited number of spurious tuples* (i.e., tuples that do not satisfy the similarity condition).

To guarantee the correctness of the join result and to reduce the number of spurious tuples, the granularity γ and the mapping function f should be carefully chosen. In fact, a too coarse granularity may cause the presence of an excessive number of spurious tuples, and a too fine granularity could cause the absence of tuples from the join result. Analogously, a bad mapping function could cause the incompleteness of the join result. (The reference point p influences neither the number of spurious tuples nor the correctness of the join). The correctness of the join result is guaranteed when the values at distance lower than α are

mapped to the same discrete value. Suppose to choose $\gamma = \alpha$, and a mapping function f:$D \rightarrow \hat{D}$ that associates with each tuple t in L and R the value v in $\hat{D}$ closest to $t[I]$ (i.e., $f(t[I])=\{v \in \hat{D} : |v - t[I]| < 0.5\gamma\}$). In this case, the equi-join result would not be correct because some tuples satisfying the original similarity condition would be omitted. For instance, consider relations CUSTOMER and PRODUCT in Figure 2(a) and assume α=5 and $p = 0$. The discrete domain $\hat{D}$ is $\{0, 5, 10, \ldots\}$ and f maps: values in the interval [0,2.5) to 0, values in the interval [2.5,7.5) to 5, and so on (we assume that the upper bound of each interval is excluded from the interval itself). Value 18 (associated with Carol) and value 14 (associated with p_4) would then be mapped to different discrete values (i.e., 20 and 15, respectively), even if the difference between them is 4. Therefore, the pair $\langle$Carol,$p_4\rangle$ satisfies the similarity condition but does not satisfy the equi-join condition over the discrete attributes. This problem happens independently from the granularity chosen. In fact, values at distance lower than α may be associated with different discrete values by function f. Consider, as an example, two values $1.5\gamma + \varepsilon$ and $1.5\gamma - \varepsilon$, with ε an arbitrarily small value, and assume that $\hat{D}$=$\{0,\gamma, 2\gamma, \ldots\}$. It is easy to see that, independently from the granularity γ, the first value is mapped by f to 2γ, while the second value is mapped by f to γ. Hence, the corresponding tuples will not satisfy the equi-join condition even if the difference between the two original values is $2\varepsilon \leq \alpha$.

Our solution consists in *replicating* the tuples in the original relations and in associating a different discrete value with each replica. The number of copies to be generated for each tuple depends on the granularity γ (i.e., the finer the granularity, the higher the number of necessary replicas). Let us consider $\gamma = \alpha$. In this case, it is necessary to duplicate each tuple in L and each tuple in R, and to associate each tuple t with the two values closest to $t[I]$ in $\hat{D}$. Hence, the mapping function f is defined as f:$D \rightarrow \hat{D} \times \hat{D}$ with $f(t[I])=\{v_1, v_2 \in \hat{D}$:$|v_1 - (t[I] - 0.5\gamma)| < 0.5\gamma$ and $|v_2 - (t[I] + 0.5\gamma)| < 0.5\gamma\}$. This approach, although effective, has the drawback of doubling the data transferred from the storage servers to the computational server. If instead $\gamma = 2\alpha$, only the tuples in one of the two relations (say R) should be duplicated. In this case, it is sufficient to associate each tuple l in L with the discrete value nearest to $l[I]$. Each tuple r in R is instead duplicated and associated with the two discrete values nearest to $r[I]$. This approach limits the communication overhead as only one of the two relations (possibly the smallest) is duplicated. Further, increasing γ does not provide advantages and causes a higher number of spurious tuples in the equi-join result. A good balance between the number of spurious tuples in the join result and the number of additional tuples in the relations is then $\gamma = 2\alpha$. Figure 2(c) illustrates how the values of attributes `Availability` and `Price` are mapped assuming $\alpha = 5$ and $p = 0$. The mapping function for relation CUSTOMER is $f_L(t[I])=\{v \in \hat{D}$:$|v - t[I]| < 5\}$, while the function for relation PRODUCT is $f_R(t[I])=\{v_1, v_2 \in \hat{D}$:$|v_1 - (t[I] - 5)| < 5$ and $|v_2 - (t[I] + 5)| < 5\}$. The original domain is then partitioned in a different way for the two relations. In particular, there is a shift of α=5, which guarantees an intersection between the intervals of original values associated with the same discrete values in L

and R, guaranteeing the effectiveness of the equi-join condition. As an example, values in [5,15) in CUSTOMER are mapped to 10, as well as the values in [0,20) in PRODUCT (intervals [0,10) and [10,20)), with an intersection of width γ=10. The relations resulting from the discretization are then formally defined as follows.

Definition 2 (Discretized tables). *Let $L(I$,Attr) and $R(I$,Attr) be two relations, I be the join attribute defined over domain D, and α be the threshold fixed by the similarity condition. The* discretized versions *$\hat{L}$ of L and $\hat{R}$ of R are two relations defined over schema $(I$,Attr,$\hat{I})$ where the domain of $\hat{I}$ is the discretized domain $\hat{D}$ of D with $\gamma = 2\alpha$, and:*

- $\forall l \in L$, $\exists \hat{l} \in \hat{L}$ *s.t.* $\hat{l}[I]=l[I]$, $\hat{l}$[Attr]=l[Attr], *and* $\hat{l}[\hat{I}]=f_L(l[I])$, *with* $f_L : D \rightarrow \hat{D}$ *and* $f_L(l[I])=\{v \in \hat{D} : |v - l[I]| < \alpha\}$;
- $\forall r \in R$, $\exists \hat{r}_1, \hat{r}_2 \in \hat{R}$ *such that* $\hat{r}_1[I]=\hat{r}_2[I]=r[I]$, $\hat{r}_1$[Attr]=$\hat{r}_2$[Attr]=r[Attr], *and* $(\hat{r}_1[\hat{I}],\hat{r}_2[\hat{I}])=f_R(l[I])$, *with* $f_R : D \rightarrow \hat{D} \times \hat{D}$ *and* $f_R(r[I])=\{v_1, v_2 \in \hat{D} : |v_1 - (r[I] - \alpha)| < \alpha$ *and* $|v_2 - (r[I] + \alpha)| < \alpha\}$.

Figure 2(b) represents the discretized version of relations CUSTOMER and PRODUCT in Figure 2(a), obtained considering the discretized domain in Figure 2(c). Note that each original tuple in PRODUCT is replaced by two tuples with discrete values representing the end-points of the interval to which the original value belongs. The size of the discretized relation is then twice as the size of the original relation.

4 Join evaluation and correctness of the approach

Like for the execution of an equi-join, the storage and computational servers do not need to coordinate for join execution. The client sends to the storage servers their sub-queries along with the information necessary to encrypt their relations, to generate markers and twins, and to perform the discretization process. The storage servers execute their sub-query and on the resulting relations apply the discretization process illustrated in Section 3. The storage servers then project attribute $\hat{I}$, insert markers and twins, and encrypt the resulting relations. The execution of the join then proceeds according to the semi-join strategy described in Section 2. Due to the discretization process, the relation $\hat{J}$, resulting from the recombination performed by the client and the integrity check and clean up phase, may include spurious tuples. In fact, the maximum distance between two (non discretized) values in D that map to the same discrete value in $\hat{D}$ is 3α (e.g., 6 in L and 19 in R are both mapped to 10). The client will then filter spurious tuples from $\hat{J}$ to obtain the approximate join result J. For instance, with reference to relations CUSTOMER and PRODUCT in Figure 2(a), consider a query that aims to return, for each customer, the products that have a price within a range of 5 with respect to what the customer is willing to spend (which is represented by attribute `Availability`). Figure 3 illustrates the evaluation of such an approximate join query with similarity condition |`Availability`−`Price`|<5, adopting the discretized domain in Figure 2(c). Relation $\hat{J}$ includes one spurious tuple combining Carol (with value 18 mapped to 20) with p_1 (with value

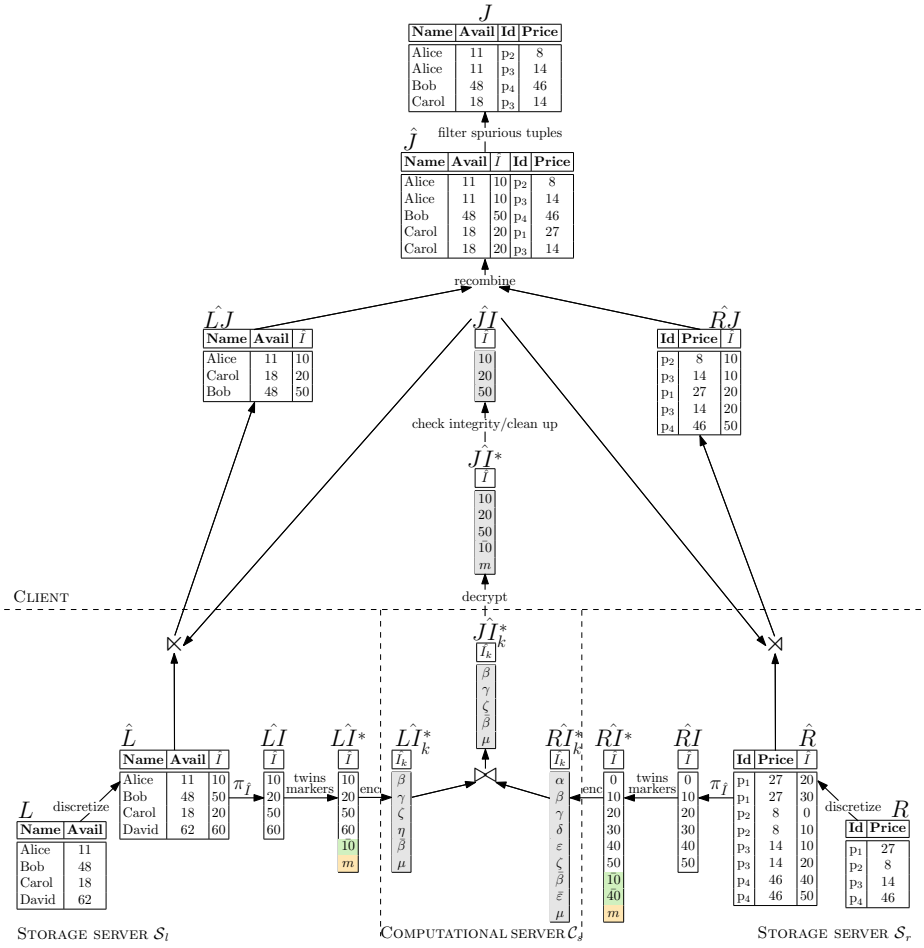


Fig. 3. An example of execution of an approximate join

27 mapped to 20 and 30) even if the difference between their values is 9 (hence greater than 5). We note that spurious tuples can be filtered from the result only when the join result has been completely reconstructed. However, this filtering can be combined with the evaluation of possible selection conditions involving attributes in both relations (i.e., condition C_{lr}) that only the client can evaluate.

The adoption of encryption on the fly, markers, and twins guarantees the correctness and completeness of the equi-join result [2]. More precisely, the probability $\wp$ that the omission of d tuples by the computational server go undetected is equal to $\wp_m \cdot \wp_t$, where $\wp_m = (1 - d/F)^m$ is the probability that no marker is omitted and $\wp_t = ((1 - d/F)^2 + (d/F)^2)^t$ is the probability of either omitting or preserving every pair of twins without detection by the client, with F the number of tuples in the join result (including m markers and t twins). We

note that, as discussed in [2], a limited number of markers and twins provide strong protection guarantees (e.g., 50 markers and 5% twins reduce to 0.007 the probability that an omission of 50 tuples goes undetected, independently from the number of tuples in the join result). To demonstrate the correctness of our approach, we only need to prove that the discretization process does not discard tuples that satisfy the approximate join condition from the equi-join result, as stated by the following theorem. (The proof has been omitted from the paper for space constraints.)

Theorem 1 (Completeness). *Let L and R be two relations, $\hat{L}$ and $\hat{R}$ be their discretized version (Definition 2). Relation $\hat{J}$ resulting from the equi-join between $\hat{L}$ and $\hat{R}$ includes all the tuples in the result of the approximate join between L and R with similarity condition $|L.I - R.I|<\alpha$.*

If the computational server behaves correctly, the equi-join result includes all the tuples of the approximate join formulated by the client and some additional spurious tuples, which can be easily identified and removed. The discretization process does not compromise data confidentiality. In fact, the computational server only receives the encrypted values of the discretized join attribute. Furthermore, the frequency distribution of discretized join values is not revealed to the computational server, because it operates on relations including the discrete join attribute only where the duplicate values have been removed by projection.

5 Experimental results

To evaluate the performance of the proposed approach, its effectiveness, and the amount of spurious tuples introduced by the discretization process, we implemented a Java prototype enforcing our protection techniques. We tested the prototype using a machine with Intel Core i5-2400, 3.10GHz CPU and 8.00GB RAM. We randomly generated between $1,000$ and $5,000$ tuples in the two relations. The join attribute values have been generated following a Zipf probability distribution with ζ between 0 and 1 (lower values of ζ correspond to more occurrences of fewer values), and with a domain including between $1,000$ and $2,500$ different values. We fixed the number of markers to 100 and the number of twins to 25% of the tuples in the original relations, which is much more than the values we expect to be used in real-world scenarios. The experimental results are computed as the average of five runs.

Spurious tuples. Figure 4 compares the percentage of spurious tuples obtained with parameter ζ of the Zipf function equal to 0.1, 0.5, and 1, varying the value of threshold α, and with relations of $1,000$ tuples (Figure 4(a)) and $5,000$ tuples (Figure 4(b)). The number of spurious tuples is not influenced by the number of tuples in the original relations but grows with α. In fact, a higher threshold implies a higher number of matching tuples in the approximate join result, but also a larger grain of discretization. The number of false positive matches then grows since the values mapped to the same discrete value becomes larger. Also

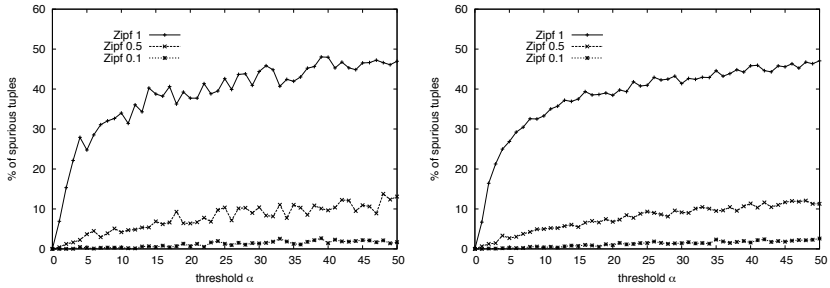


Fig. 4. Percentage of spurious tuples in the equi-join result varying threshold α and the Zipf parameter ζ, with relations including $1,000$ (a) and $5,000$ (b) tuples

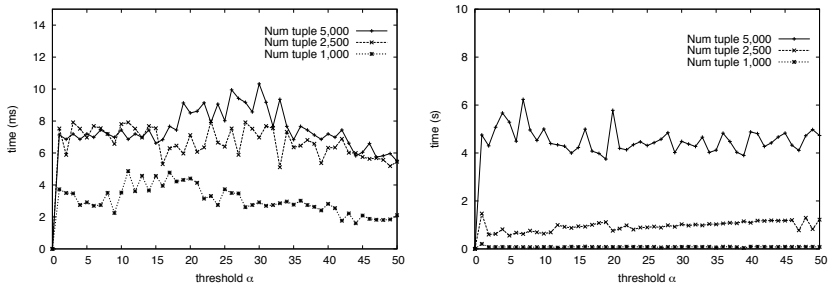
the distribution of the frequency of the join attribute values influences the percentage of spurious tuples. Figure 4 shows that the percentage of spurious tuples is always below 5% when ζ is 0.1, and below 15% when ζ is 0.5. The percentage grows when ζ is 1 and, for high values of α, reaches 45%.

Response time. A second set of experiments was aimed at analyzing the response time of an approximate join query. We focused on the overhead caused by the discretization and filtering processes, which are specific of the translation of an approximate join into an equi-join. We considered configurations characterized by relations of different sizes, generated in such a way that the distribution of the join attribute values follow a Zipf distribution with parameter $\zeta = 0.5$.

Figure 5(a) illustrates the time required for the discretization process, which takes place at the storage servers, varying threshold α. The figure compares the values obtained considering relations of three different sizes. As expected, the discretization time grows with the size of the relations. In fact, the storage server needs to associate one (or two) discrete value(s) with each tuple in its relation. The discretization time is instead not affected by the value of α since the computation of the discrete values does not depend on the granularity of the discretized domain. It is interesting to note that the time necessary for the discretization process is always very low (less than 10ms).

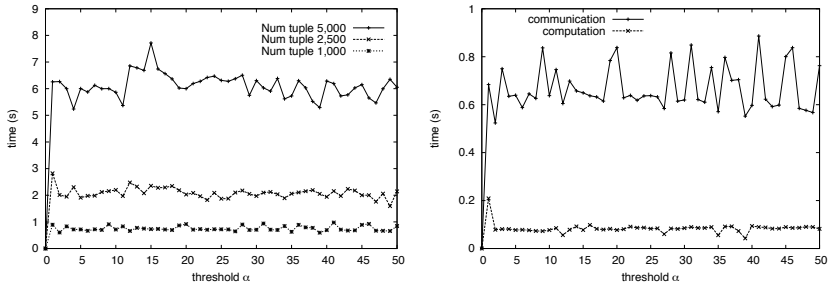
Figure 5(b) reports the time for the client to filter spurious tuples, varying threshold α and comparing three configurations obtained with relations of different sizes. Like for the discretization process, the time necessary for filtering spurious tuples does not depend on α, but it depends on the number of tuples in the relations, and then also in the join result. In fact, the client needs to check every tuple in the join result to discard spurious tuples. The overhead caused by filtering is however limited, remaining below 7s even for relations with $5,000$ tuples (less than 0.05s for relations with $1,000$ tuples).

The adoption of a semi-join, in contrast to a regular join, strategy for query evaluation implies an additional overhead for the client due to the recombination of the join result computed over the join attribute with the semi-tuples



(a) (b)

Fig. 5. Time taken by the discretization process (a) and by the filtering process (b) varying α and the number of tuples in the relations



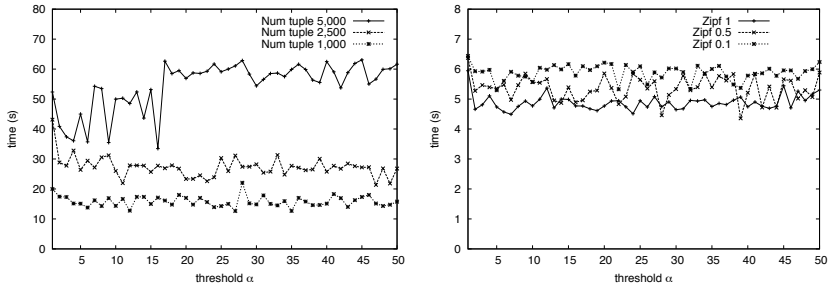
(a) (b)

Fig. 6. Time taken by the client to recombine the join result (a) and its computation and communication components (b), varying α and the number of tuples in the relations

received from the storage servers. Figure 6(a) illustrates the overhead of the recombination phase, obtained summing the communication time of sending the semi-tuples to the client and the computation time for the client to obtain the final result. As expected, the recombination time grows with the size of the join result, but it is not affected by the discretization threshold α. Figure 6(b) illustrates the communication and computation components of the recombination overhead obtained with relations of $1,000$ tuples. As expected, the communication time is higher than the computation time.

Figure 7(a) compares the (total) response time for the computation of an approximate join of configurations obtained varying the number of tuples in the original relations. The response time is higher for relations with a higher number of tuples, and does not depend on α. Figure 7(b) compares the response times obtained joining relations with 1,000 tuples each, but generated with different values for the parameter ζ of the Zipf distribution. We can observe that the response time is not affected by this parameter.

To better assess the impact of the discretization process in the computation of an approximate join result, we analyzed the impact of each component of



(a) (b)

Fig. 7. Overall response time varying the number of tuples in the relations (a) and the parameter ζ of the Zipf distribution (b)

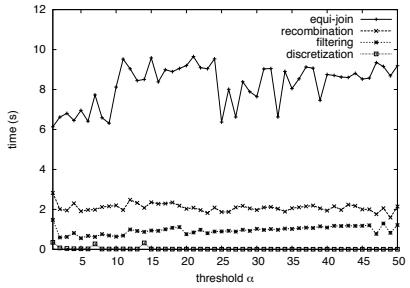


Fig. 8. Components contributing to the response time

the response time. Figure 8 illustrates the contribution to the total response time due to each phase of the process. The figure shows that the discretization time has a very limited impact (0.27% on average), as well as the filtering phase (nearly 11.20% on average). The time necessary for the recombination is higher (22.52% on average), but it also includes communication costs. However, the most time consuming phase is the evaluation of the equi-join (nearly 66.02% on average of the response time) and is delegated to the computational server. We can then conclude that also approximate joins can benefit from the presence of inexpensive external computational servers (especially if threshold α is low).

6 Related work

Previous related work has been devoted to protect the confidentiality of data outsourced to *honest-but-curious* servers (e.g., [8,15,16]). Most of these solutions encrypt data before outsourcing and complement them with indexes designed to support different kinds of SQL clauses (e.g., [5,8]).

Other works have considered the problem of guaranteeing integrity when the external server is not trusted. These solutions are based on the adoption of authenticated data structures or on probabilistic approaches. Approaches that rely

on authenticated data structures (e.g., Merkle trees [12] and signature-based schemas [13]) return, together with the query result, a verification object that is used by the client to verify the correctness and completeness of the result. Authenticated data structures provide deterministic guarantees but they are defined over a specific attribute and only queries operating on it can be verified. Probabilistic approaches can be adopted with any query, but provide probabilistic guarantees only (e.g., [3,17,18]). The approach in [18] inserts into the original relation a set of fake tuples, generated according to a deterministic function, before outsourcing the relation. Absence of the expected fake tuples in a query result signals its incompleteness. The solution in [17] duplicates a subset of the tuples in the original relation and encrypts them with a different key. Since the external server cannot recognize duplicated tuples, their absence from the query result signals a misbehavior. The use of twins and markers for the join integrity verification has been first introduced in [2,3,4]. Here, we extend these proposals to the support of approximate joins. Besides correctness and completeness, techniques aimed at providing freshness by periodically changing the verification object have also been proposed (e.g., [19]).

A related, but different, line of work is represented by discretization approaches. The solutions proposed for producing a discrete version of continuous domains have the goal of making data suitable to machine learning and/or data mining applications, of supporting proximity tests, or of anonymizing pseudonyms (e.g., [6,9,11,14]). These solutions are therefore not suited to the scenario considered in this paper. The goal of works studying the evaluation of approximate joins is to limit the performance impact due to the evaluation of conditions based on distance measures (e.g., [1,7]). These solutions cannot then be adopted in our scenario as they do not not operate over encrypted data and hence do not translate approximate into equality conditions.

7 Conclusions

We have presented an approach that enables a user to assess the integrity of the result of an approximate join query, leveraging on the techniques introduced for equi-join queries. We have proposed a discretization of the join attribute to translate an approximate join into an equi-join query. Due to the discretization process, the join result may include additional (spurious) tuples that the client must remove. Also, the experimental evaluation has confirmed the effectiveness of our approach and has demonstrated its limited overhead. Our work leaves space to further investigations, including the consideration of non-Euclidean distance metrics, possibly also operating in multidimensional scenarios.

Acknowledgements. The authors would like to thank Riccardo Moretti for support in the implementation of the system and in the experimental evaluation. This work was supported in part by: the EC within the 7FP under grant agreement 312797 (ABC4EU) and within the H2020 under grant agree-

ment 644579 (ESCUDO-CLOUD), the Italian Ministry of Research within PRIN project "GenData 2020" (2010RTFWBH), and NSF under grant IIP-1266147

References

1. Das, A., Gehrke, J., Riedewald, M.: Approximate join processing over data streams. In: Proc. of ACM SIGMOD. San Diego, CA (June 2003)
2. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., Samarati, P.: Integrity for distributed queries. In: Proc. of CNS. San Francisco, CA (October 2014)
3. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Integrity for join queries in the cloud. IEEE TCC 1(2), 187–200 (2013)
4. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Optimizing integrity checks for join queries in the cloud. In: Proc. of DBSec. Vienna, Austria (July 2014)
5. De Capitani di Vimercati, S., Foresti, S., Samarati, P.: Managing and accessing data in the cloud: Privacy risks and approaches. In: Proc. of CRiSIS. Cork, Ireland (October 2012)
6. Dougherty, J., Kohavi, R., Sahami, M.: Supervised and unsupervised discretization of continuous features. In: Proc. of ICML. San Francisco, CA (July 1995)
7. Gravano, L., Ipeirotis, P., Jagadish, H., Koudas, N., Muthukrishnan, S., Srivastava, D.: Approximate string joins in a database (almost) for free. In: Proc. of VLDB. Rome, Italy (September 2001)
8. Hacigümüş, H., Iyer, B., Mehrotra, S., Li, C.: Executing SQL over encrypted data in the database-service-provider model. In: Proc. of SIGMOD. Madison, WI (June 2002)
9. Han, J., Kamber, M.: Data Mining, Southeast Asia Edition: Concepts and Techniques. Morgan Kaufmann (2006)
10. Jhawar, R., Piuri, V., Samarati, P.: Supporting security requirements for resource management in cloud computing. In: Proc. of CSE. CSE 2012, Paphos, Cyprus (December 2012)
11. Kerschbaum, F.: Distance-preserving pseudonymization for timestamps and spatial data. In: Proc. of WPES. Alexandria, VA (October 2007)
12. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Authenticated index structures for aggregation queries. ACM TISSEC 13(4), 32:1–32:35 (2010)
13. Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. ACM TOS 2(2), 107–138 (2006)
14. Nielsen, J., Pagter, J., Stausholm, M.: Location privacy via private proximity testing. In: NDSS. San Diego, CA (February 2011)
15. Ren, K., Wang, C., Wang, Q.: Security challenges for the public cloud. IEEE Internet Computing 16(1), 69–73 (2012)
16. Samarati, P., De Capitani di Vimercati, S.: Data protection in outsourcing scenarios: Issues and directions. In: Proc. of ASIACCS. Beijing, China (April 2010)
17. Wang, H., Yin, J., Perng, C., Yu, P.: Dual encryption for query integrity assurance. In: Proc. of CIKM. Napa Valley, CA (October 2008)
18. Xie, M., Wang, H., Yin, J., Meng, X.: Integrity auditing of outsourced data. In: Proc. of VLDB. Vienna, Austria (September 2007)
19. Xie, M., Wang, H., Yin, J., Meng, X.: Providing freshness guarantees for outsourced databases. In: Proc. of EDBT. Nantes, France (March 2008)