Managing distributed queries under anonymity constraints

Axel Michel
SDS team at LIFO
INSA Centre Val de Loire
Bourges, France
axel.michel@insa-cvl.fr

Benjamin Nguyen
SDS team at LIFO
INSA Centre Val de Loire
Bourges, France
benjamin.nguyen@insa-cvl.fr

ABSTRACT

In this paper, we consider the problem of the collection of microdata from users, in order to compute aggregate queries. Our main focus is on matching anonymity requirements between users and queriers. Our work is based on the Trusted Cells architecture and SQL/AA secure distributed query evaluation framework.

1. INTRODUCTION

In many scientific fields, ranging from medicine to sociology, computing statistics on personal information is central to the discipline's methodology. With the advent of the Web, and the massive databases that compose it, statistics have become "data science": turning large volumes of microdata into knowledge. The general public is slowly getting aware that their own personal data can be unwillingly shared with applications processing it, with many different objectives. Many are of obvious use to the community, such as using medical data to improve the knowledge of diseases, or sharing energy consumption in smart grids. In all these applications, the knowledge comes from analysing aggregated data. In most cases, these operations (*i.e.* global database queries) provide equivalent results when run on anonymized data.

In this article, we look at the specific problem of users participating in global queries, while respecting their anonymity constraints (*e.g.* k-anonymity as defined by Sweeney in [5]). We build upon the *Trusted Cells (TC)* [1] vision (see Section 2), an *asymmetric architecture (AA)* based mixing a large number of small, personal and inexpensive trusted devices which equip individual participants (called *Trusted Data Servers* or TDS) and an untrusted cloud infrastructure. In this architecture, each user keeps their microdata stored in a local, secured, relational database, called *PlugDB*[1].

The *SQL/AA* [6] system (see Section 2.2) is the distributed query engine running on the *asymmetric architecture* which can compute aggregate queries over user data stored in the millions of individual TDSs without leaking any microdata during the query evaluation process.

However, the question of how user participation is managed in *SQL/AA* has not been studied. Thus our contributions, presented in Section 3 in this article concern the definition of user anonymity requirements, and the evaluation of these requirements, when deciding whether to participate in a global aggregate query or not. Necessary background information is presented in Section 2 and Section 4 concludes.

1. See: https://project.inria.fr/plugdb/en/

2. BACKGROUND AND PROBLEM STATEMENT

Our work builds upon the Trusted Cells asymmetric architecture, presented in 2010 [1]. The purpose of this architecture is to preserve users privacy by preventing data leaks during computations on their data. Our focus here is on the computation of SQL aggregations.

2.1 The Trusted Cells asymmetric architecture

Computations managed through the collaboration of two parties. The first party is a (potentially large) set of **Trusted Data Servers** (TDSs) which are tamper resistant hardware (*e.g.* set-top box, secure USB token, smart phone). They are (1) **trusted** but have (2) **limited resources** and (3) **low availability**. Characteristics (2) and (3) impose another party to compute expensive operations (*e.g.* collecting, sorting), the **Supporting Server Infrastructure** (SSI). It is an (1) **untrusted** infrastructure (*e.g.* the Cloud) with (2) **high performances** and (3) **high availability**. The **threat model** takes place because of the nature of the SSI. It considers two kinds of SSI, **honest-but-curious**, where the SSI tries to infer users'personal data but does not modify protocol, and **malicious**, where the SSI may tamper the protocol to infer data.

2.2 SQL/AA: global queries on Trusted Cells

SQL/AA is a protocol to execute SQL on the Trusted Cells architecture [6]. Once an SQL query is issued by a querier, it is computed in three phases: first the *collection phase* where the TDSs decide to participate or not (*i.e.* send dummies in that case) in the computation, evaluate the `WHERE` clause and return encrypted data to the SSI. Second, the *aggregation phase*, where TDSs receive encrypted data from the SSI, decrypt it, and compute aggregations (*e.g.* `AVG`, `COUNT`). Finally the *filtering phase*, where TDSs produce the final result by filtering out the `HAVING` clause and send the result to the querier. We have chosen to use *SQL/AA* as query execution infrastructure in our work, because of its capacities to securely compute relational queries (*i.e.* without leaking any information).

2.3 Anonymity

In order to protect the privacy of users, queries must respect a certain degree of anonymity. Many anonymity models exist, such as *differential privacy* [2], $k-$anonymity [5] or ℓ-diversity [4]. In this article, we consider those two last models, k-anonymity and ℓ-diversity. A table is $k-$anonymous if and only if each distinct non-sensitive value is present

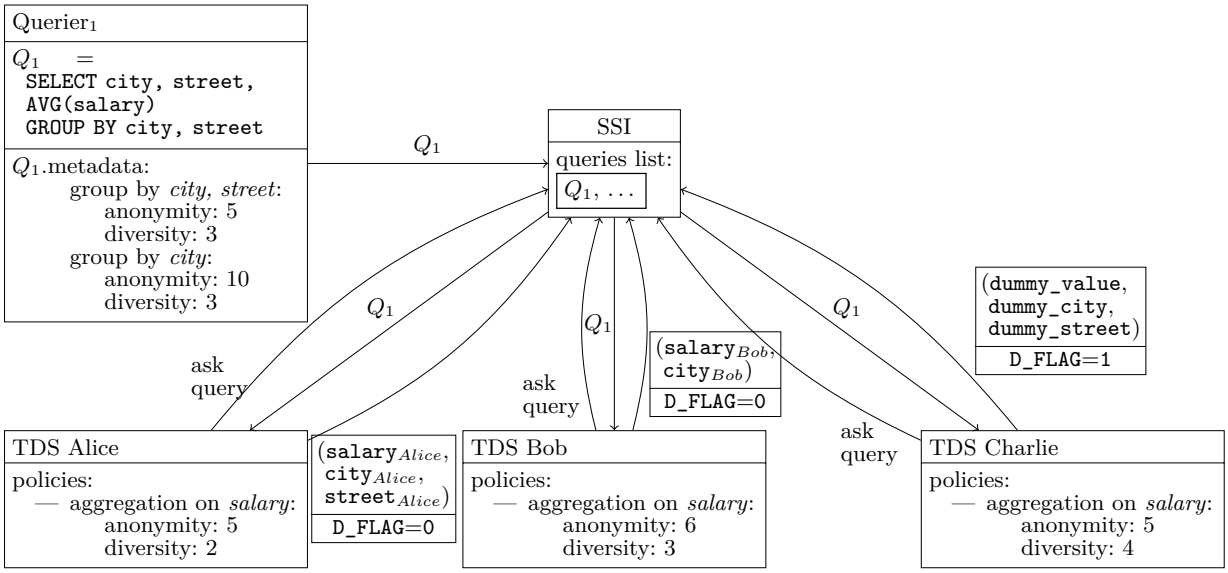

Figure 1: Example of collection phase with anonymity constraints.

at least k times (*e.g.* table 1). A $k-$anonymous table is $\ell-$diverse if and only if each group of the same non-sensitive value has at least ℓ distinct sensitive values (*e.g.* table 1). We model those using `GROUP BY` and `HAVING` clauses in SQL (see Section 3.1).

Non-sensitive		Sensitive
ZIP	Age	Disease
112**	> 25	Cancer
112**	> 25	Cancer
112**	> 25	Heart Disease
1125*	*	Heart Disease
1125*	*	Viral Infection
1125*	*	Cancer

Table 1: 3−anonymisous and 2−diverse table

2.4 Problem Statement

In this article, we investigate how to run queries respecting anonymity constraints defined by users on SQL/AA. Anonymity requirements must be defined by (a) the querier on her query and (b) users on their microdata. The problem statement is justified by the necessity to provide privacy-preserving and more control to users' data.

3. ENFORCING ANONYMITY IN SQL/AA

Consider the simple SQL query Q_1 computing the average salary of users grouped by city, street (see Fig. 1).

Assume that *Alice* is a TDS user. She does not know how many other users are on her street. She may not want to give her data if there are less than five TDS users in her group (*i.e.* 5−anonymity). This condition can be enforced using standard SQL as presented next.

3.1 Modelling anonymisation using SQL

General Principle: The generalisation of data is a high cost operation that TDSs can't compute easily. To avoid this operation, a querier can declare an anonymity guarantee in the meta-data section of her query. TDSs can send their real data (or dummies) if guarantees are good enough by comparing those with anonymity constraints declared by users. Anonymity guarantees are ensured by the SQL/AA system. It is important to note that this can be implemented by adding the following clause to the query: `HAVING COUNT(*)>=5`.

Alice, who is willing to share her salary data with a query enforcing 5-anonymity will thus know that she can participate in Q_1 by inspecting $Q_1.metadata$. Thus, she will encrypt her data and a tag indicating that the data is real, and send it back to the SSI (as in the protocol shown in [6]). Consider Bob, who is only willing to participate if 6-anonymity is enforced. He will not send his real data, but instead send fake data, and a tag indicating this data is fake (both information is encrypted), called a *dummy tuple*. Note that Bob *must* send some answer to the SSI, otherwise it would be able to infer some information about Bob's privacy policy.

Enforcing ℓ-diversity: to ensure $\ell - diversity$ with $\ell > 1$, the clause `COUNT(DISTINCT salary)` can be used. Since this clause is an holistic function, we can compute it while the *aggregation phase* by adding naively each distinct value under a list or using a cardinality estimation algorithm such as *HyperLogLog* [3].

Defining user anonymity policies: Each user defines a set of policies, composed of sensitive attributes and anonymity parameters. These policies are stored inside the TDS's database and are private.

Comparing queries with user anonymity constraints: Each TDS rewrites the global query using its own privacy views. If a TDS is not able to rewrite the query, or if the query does not provide any results, then two possibilities occur. If it is possible to rewrite the query locally by generalising it (e.g. replacing `group by city, street` by `group by city`) and if the query accepts this generalisation scheme (see $Q_1.metadata$ in Fig. 1) then this generalised answer is produced. This is what happens to Bob: answering the `group`

city	street	AVG(salary)	COUNT(∗)	COUNT (DISTINCT salary)
Le Chesnay	Dom. Voluceau	1500	6	4
Le Chesnay	******	1700	12	7
Bourges	Bv. Lahitolle	1600	3	3
Bourges	******	1400	11	7

Table 2: Filtering phase pre-result example.

`by city, street` clause is not acceptable, but answering just with `city` is. Otherwise (e.g. in the case of Charlie), dummy tuples will be generated.

3.2 The modified protocol

Collection phase: After TDSs download the query, they compare their anonymity constraints with the query anonymity guarantees. As shown above (see section 3.1) if anonymity constraints can't be satisfied, users send dummy tuples. In the case of real data are sent, the difference between generalised query and the initial query is the detail level of the `GROUP BY` attribute.

Aggregation phase: To ensure that anonymity and diversity can be verified, clauses `COUNT(*)` and `COUNT(DISTINCT salary)` are computed in addition of the aggregation asked by the querier. Generalised and not generalised group are differentiate and their aggregations are not computed together.

Filtering phase: If `HAVING` clauses can't satisfy the privacy guarantees then non generalised groups will be added into the generalised group they belong. In the example table 2 the tuple `(Bourges, Bv.Lahitolle, 1600)` is merged with the tuple `(Bourges, 1400)` to form the tuple `(Bourges, 1442.86)`.

4. CONCLUSION

In this paper, we presented our approach to define and enforce users' anonymity policies on SQL `GROUP BY` queries using the Trusted Cells architecture. Future work will include large scale testing of the efficiency of this approach.

5. REFERENCES

[1] T. Allard, N. Anciaux, L. Bouganim, Y. Guo, L. L. Folgoc, B. Nguyen, P. Pucheral, I. Ray, I. Ray, and S. Yin. Secure personal data servers: a vision paper. *PVLDB*, 3(1):25–35, 2010.

[2] C. Dwork. Differential privacy. In *Proceeding of the 39th International Colloquium on Automata, Languages and Programming*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin / Heidelberg, 2006.

[3] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *Proceedings of the 2007 International conference on Analysis of Algorithms (AOFA'07)*, 2007.

[4] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 24, 2006.

[5] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[6] Q. To, B. Nguyen, and P. Pucheral. SQL/AA: executing SQL on an asymmetric architecture. *PVLDB*, 7(13):1625–1628, 2014.