Multi-Provider Secure Processing of Sensors Data

Enrico Bacis*, Sabrina De Capitani di Vimercati†, Dario Facchinetti*, Sara Foresti†,
Giovanni Livraga†, Stefano Paraboschi*, Marco Rosa*, Pierangela Samarati†
* Università degli Studi di Bergamo, Italy – Email: *name.surname*@unibg.it
† Università degli Studi di Milano, Italy – Email: *name.surname*@unimi.it

***Abstract*—We describe the implementation of an approach for supporting secure query processing over sensors data in a multi-provider scenario. Our solution relies on the definition of authorizations regulating access to data according to three different visibility levels (no visibility, encrypted visibility, and plaintext visibility). Data processing is performed by multiple providers based on the restrictions imposed by authorizations, which may require to adjust data visibility on the fly. We describe the structure of the query optimizer and show how the operations of a computation can be assigned to different cloud providers to build an efficient, secure, and economical plan for collaborative data processing.**

I. Introduction

Sensors pervasiveness makes it possible to collect huge amounts of data for the most diverse services and uses. The management of such data requires to address several challenges related to their storage, sharing, and processing [1]. In many cases, it is critical to perform complex analysis, possibly integrating data coming from different sources, in an efficient and secure way. In fact, sensors may generate sensitive information [2] that not all parties can access and may also be subject to law restrictions, such as those imposed by the EU General Data Protection Regulation (GDPR). The availability of modern and flexible distributed frameworks (e.g., Apache Spark [3]) permits to build efficient solutions able to manage large data collections in multiple formats and under the control of different authorities, using tools compatible with the structure of cloud architectures.

In this paper, we address the problem of supporting a secure and efficient query processing over data collected and controlled by different parties (data authorities). In particular, our reference scenario (Figure 1) is characterized by multiple data authorities collecting data from sensors that aim to collaborate for the evaluation of queries over their data, possibly involving cloud providers that offer computing power at limited prices but that are not necessarily fully trusted to access the data content [4]. Our approach allows the data authorities to independently define authorizations over their data regulating their release to users, other data authorities, and cloud providers. Such authorizations can grant plaintext or encrypted visibility over the data. Query processing can then involve different data authorities and/or cloud providers, according to authorizations, to minimize economic costs or maximize performance. On-the-fly encryption/decryption is applied to allow or to block data visibility as required by the authorizations and the operation requirements. To perform complex data analysis, our approach supports the integration of *User Defined Functions* (UDFs) with traditional SQL queries. UDFs are modeled as black boxes that correspond to procedural computations constructed using a variety of programming languages and paradigms.

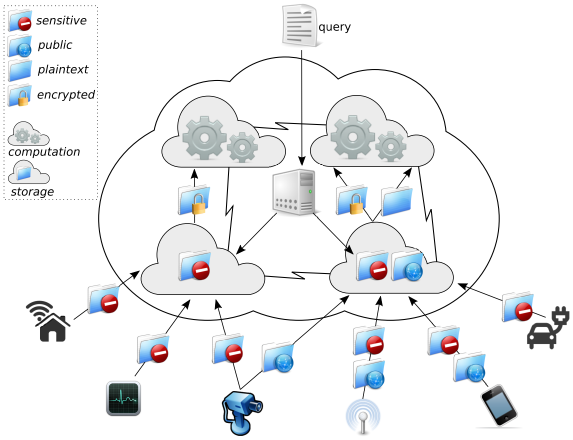


Fig. 1. Reference scenario

In the remainder of this paper, we describe the query optimizer designed for manipulating and transforming query plans to satisfy the authorizations and to apply the correct protection technique. We design a two-step cost optimizer that is easily pluggable within a real query optimization chain. Our implementation demonstrates the advantages of collaborative query execution in the open cloud market and shows the effectiveness of a two-step query optimization approach, which integrates security requirements while balancing the economic cost of query execution and optimization times. With our solution, an organization can take advantage of inexpensive computing power to bring down the costs and increase the flexibility and scalability of data analysis in sensor networks.

II. Reference Model

We assume that the data collected and controlled by data authorities are organized in relational tables and that computations and analysis over these data can be expressed as SQL queries, extended to support arbitrary UDFs. Figure 2(a) illustrates an example of two relation schema storing information generated by a portable ECG monitor and a fit tracker. To enable data authorities to specify which subject (i.e., data

This work was supported by the EC within the H2020 Program under grant agreement 825333.

HEART(`Date`, `ECG`, `Alert`) ; ACTIVITY(`Day`, `Steps`)

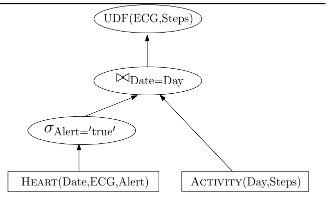
(a) Relations schema

SELECT UDF(`ECG`, `Steps`)
FROM HEART JOIN ACTIVITY
ON `Date` = `Day`
WHERE `Alert` = 'true'

(b) Query

[{`Date`}, {`ECG`}]	→ *Amazon*
[_, {`Day`, `Steps`}]	→ *Amazon*
[{`Date`}, {`ECG`, `Alert`}]	→ *Google*
[{`Day`}, {`Steps`}]	→ *Google*

(c) Authorizations



(d) Query tree plan

Fig. 2. Running example

authority, cloud provider, final user) can access their data, in plaintext or in encrypted form, we rely on the authorization model in [4]. According to this model, an *authorization* $[P, E] \rightarrow S$ over relation R allows subject S to access the set P of attributes in plaintext and the set E of attributes in encrypted form. S is not authorized for all the other attributes in R. For instance, authorization [{`Date`}, {`ECG`}]→ *Amazon* in Figure 2(c) states that *Amazon* has plaintext visibility over `Date`, encrypted visibility over `ECG`, and no visibility over `Alert`.

Each relation (base or resulting from the evaluation of a subquery) is associated with a *profile* modeling its information content. The profile of a relation includes plaintext attributes, encrypted attributes, and equivalence relationships among attributes (i.e., attributes involved in a condition comparing them). Indeed, the values of attributes compared in query evaluation are related in the resulting relation, and are all exposed even if the schema includes only one of them. Note that a relation profile keeps track of attributes represented both explicitly (i.e., appearing in the schema) and implicitly (i.e., involved in conditions or grouping) in the relation. For instance, consider the query in Figure 2(b), the relation profile of the result includes `ECG` and `Steps`, explicitly represented in the relation schema, but also `Alert` and the equivalence between `Date` and `Day`.

A subject S is authorized to access a relation R iff S has: *i)* plaintext visibility over (visible and implicit) plaintext attributes in R's profile; *ii)* plaintext or encrypted visibility over (visible and implicit) encrypted attributes in R's profile; *iii)* uniform visibility over equivalent attributes (i.e., either plaintext or encrypted visibility over all equivalent attributes). Given a query formulated by a final user and its query tree plan (e.g., see Figure 2(d)), it is necessary to assign each operation in the tree to a subject respecting the authorizations, while maximizing performance and minimizing economic costs. Note that, as illustrated in [4], encryption can be profitably used to enable the assignment of operations to less expensive, but also less trusted, cloud providers. For instance, considering the example in Figure 2, encryption of attribute `ECG` would enable *Google* to evaluate the selection condition. The working of query optimizers, traditionally designed to maximize performance [5]–[7] needs then to be revised to enable the enforcement of authorizations, the injection of encryption/decryption operations in the query evaluation plan, and the minimization of economic costs (see Section III).

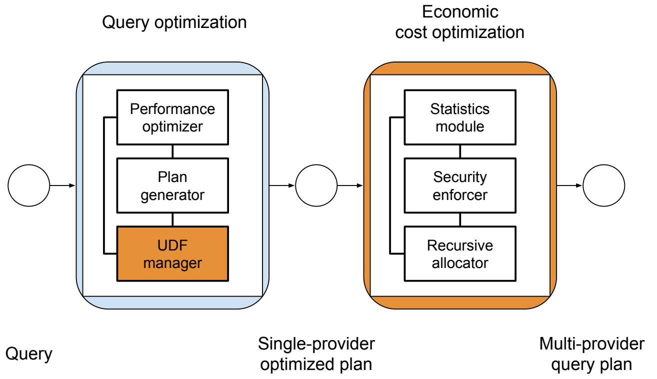


Fig. 3. Two-phase optimization process: *i)* single-provider optimizer (light blue), *ii)* economic cost optimizer (orange)

III. Query Optimizer: Implementation

Given a query, we aim at generating a query plan that minimizes economic costs. Building on the Apache Spark SQL [8] optimizer, we propose a solution based on a two-phase approach (Figure 3). Intuitively, in the first phase the optimizer computes an optimal plan assuming the presence of a single provider, authorized for plaintext visibility over all base relations. The second phase is dedicated to economic costs optimization, by assigning operations to cloud providers, possibly introducing encryption/decryption operations to guarantee authorization enforcement. The estimation of the economic cost takes into consideration relational operations/UDFs execution, data encryption, and data transfer. The main advantage of our approach, compared to solutions aimed to integrate authorization enforcement and cost optimization in existing query optimizers, is that it can be easily integrated with existing DBMSs without the need to redesign their internal architecture and modules.

A. Cost optimizer

The economic cost optimizer first performs preliminary checks (e.g., on attribute names) and builds an *oracle* plan, by extending the single-provider plan with UDF time statistics and authorizations. The structure of the oracle is then cloned, to keep track of assignment attempts. Indeed, for each operator in the query plan, the economic optimizer aims at identifying the best candidate among the available cloud providers, that is, the provider minimizing economic costs of query evaluation. Our solution for the design of an efficient economic cost optimizer adopts a greedy approach, choosing for each operation the candidate that minimizes the evaluation cost of the considered operation. Such an approach identifies a good, even if not always optimal, assignment of operations to cloud providers. The assignment process performs a post order visit of the query tree plan and, for each operation, it performs the following four steps (see the state machine in Figure 4).

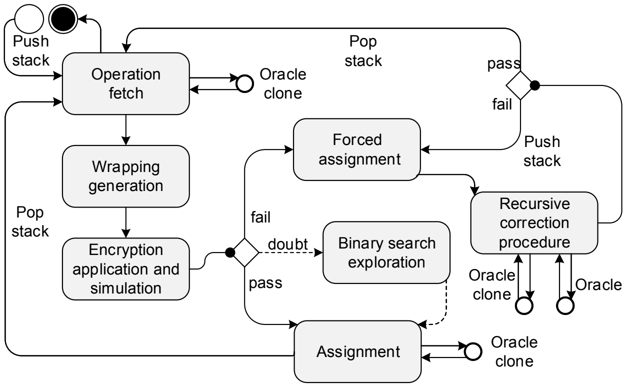


Fig. 4. State machine description of the operators assignment algorithm

1) *Identify valid candidates*: the algorithm identifies the set of cloud providers that satisfy both the authorizations and the execution needs, assuming the possibility to inject encryption and decryption operations.
2) *Estimate economic costs*: the algorithm simulates the operation execution by the candidate and generates the profile of the resulting relation, extended with an estimate of the economic cost.
3) *Check uniform visibility*: since some operators compare or combine attributes, the algorithm verifies the authorization policy also a posteriori, to check whether uniform visibility is satisfied [4], and possibly restricts the set of valid candidates.
4) *Assign candidate*: the algorithm, according to the greedy approach, assigns the operation to the valid candidate with the lowest economic cost.

Note that the algorithm does not generate an assignment with economic cost greater than the cost of the single-provider plan. A recursive validation and cost correction procedure is triggered when the assignment strategy fails to identify a multi-provider strategy with lower cost than the single-provider one.

B. Prototype

We developed a prototype to demonstrate the effectiveness of the proposed cost optimization method described above[1].

The required input files to run the optimizer are:

- `Query.xml`: the single-provider plan produced by the query optimizer;
- `DB.xml`: the database schema;
- `Providers.xml`: the authorizations and the economic costs of cloud providers.

The result produced by our optimizer is a multi-provider plan that minimizes the cost (accordingly to the greedy strategy), while satisfying authorizations. The result also specifies the encryption and decryption operations extending the original query plan. The tool currently understands relational algebra and custom UDF operators, and supports the following three operation modes: `debug_mode`, which produces a full stack report of execution, `depth_mode`, which performs an exhaustive search over the space of alternatives by setting up the maximum height of a binary prefix tree, `no_uvr`, which removes the uniform visibility condition from authorization enforcement.

TABLE I
COST ESTIMATE FOR EACH UDF COMPLEXITY, FOR EACH MODE

UDF complexity	**Single-P**	**Multi-P**	**Multi-P `no_uvr`**
linear	0.041$	0.041$ (0.0%)	0.022$ (53.7%)
pseudo-linear	0.047$	0.019$ (40.4%)	0.019$ (40.4%)
quadratic	3.465$	3.465$ (0.0%)	0.497$ (14.3%)

C. Experimental results

Since there is no industrial benchmark that models a hybrid workload (i.e., no TPC benchmark includes UDFs) we created a set of demo plans (see the repository for full details). The demo set includes queries with three types of UDF complexity: linear, pseudo-linear, and quadratic.

The tests have been executed on an Intel i5 server with 16 GB memory and SSD drive running Ubuntu 18.04 LTS. A summary of the computed economic costs for query evaluation is shown in Table I. The average cost optimization time is 26.9ms, which is compatible with the needs of most applications.

IV. CONCLUSIONS

We have described the implementation of a two-steps query optimizer for supporting secure and efficient query processing in a multi-provider scenario. The experimental evaluation of our approach has shown its effectiveness, especially for computations of high complexity. Our work, therefore, shows how to support efficient large-scale analysis by taking advantage of economically convenient cloud provider.

REFERENCES

[1] M. Jahan, S. Seneviratne, B. Chu, A. Seneviratne, and S. Jha, "Privacy preserving data access scheme for iot devices," in *Proc. of IEEE NCA*, Cambridge, MA, USA, 2017.

[2] D. Christin, A. Reinhardt, S. S. Kanhere, and M. Hollick, "A survey on privacy in mobile participatory sensing applications," *Journal of systems and software*, vol. 84, no. 11, pp. 1928–1946, 2011.

[3] "Apache spark," https://spark.apache.org/, accessed: 2018-11-26.

[4] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati, "An authorization model for multi provider queries," *PVLDB*, vol. 11, no. 3, pp. 256–268, 2017.

[5] A. Rheinländer, U. Leser, and G. Graefe, "Optimization of complex dataflows with user-defined functions," *ACM CSUR*, vol. 50, no. 3, p. 38, 2017.

[6] G. Graefe, "Volcano – an extensible and parallel query evaluation system," *IEEE TKDE*, vol. 6, no. 1, pp. 120–135, 1994.

[7] M. A. Soliman, L. Antova, V. Raghavan, A. El-Helw, Z. Gu, E. Shen, G. C. Caragea, C. Garcia-Alvarado, F. Rahman, M. Petropoulos *et al.*, "Orca: a modular query optimizer architecture for big data," in *Proc. of ACM SIGMOD*, Snowbird, UT, USA, 2014.

[8] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark sql: Relational data processing in spark," in *Proc. of ACM SIGMOD*, Melbourne, VIC, Australia, 2015.

[1] The code is publicly available at https://github.com/unibg-seclab/query-opt