Multi-Provider Secure Processing of Sensors Data

Enrico Bacis[1], Sabrina De Capitani di Vimercati[2], Dario Facchinetti[1], Sara Foresti[2], Giovanni Livraga[2], Stefano Paraboschi[1], Marco Rosa[1], Pierangela Samarati[2]
[1]Università degli Studi di Bergamo (Italy), [2]Università degli Studi di Milano (Italy)

IEEE International Conference on Pervasive Computing and Communications PERCOM 2019

Scenario

- Wide **sensors networks** collect huge amount of **data**
- Unlike gathering phase, data **processing** is not performed in close proximity to the sensor, so the data are usually sent to the cloud
- Data generated by sensors may be **sensitive** or be subject to **law restrictions**, for example the ones imposed by the General Data Protection Regulation
- Many **cloud** platforms are available, each with different **cost** and **confidentiality** profiles

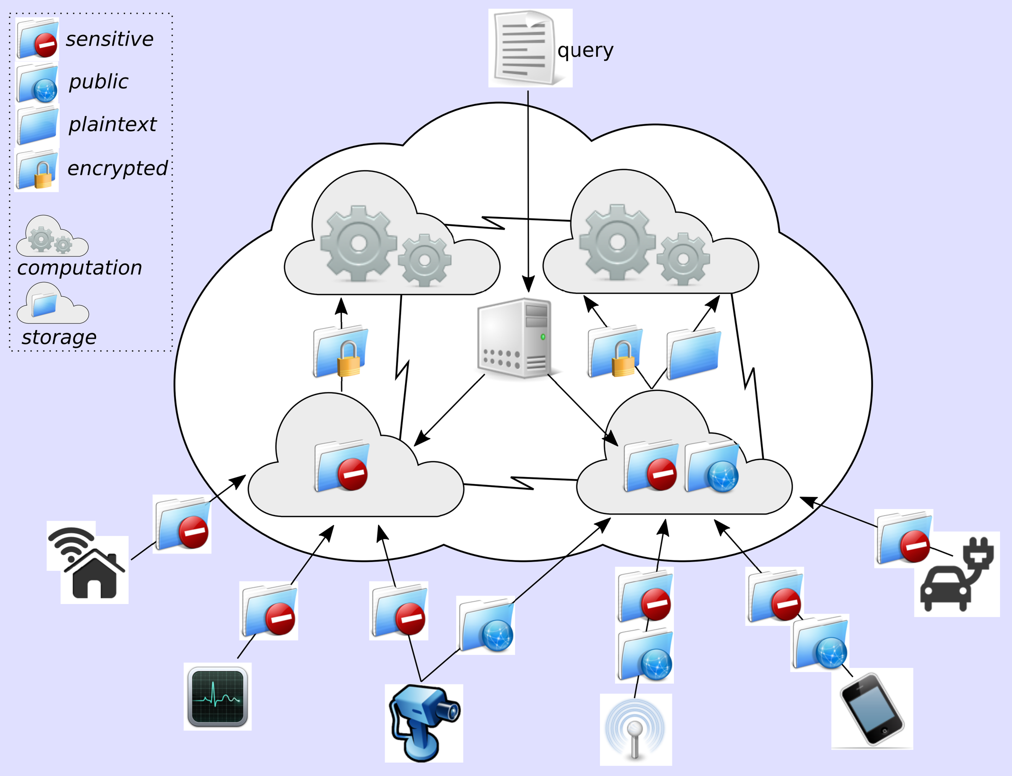


Figure: Reference scenario

Problem

Is it possible to create **collaborative multi-provider query plans**, leveraging the benefits of open cloud market, while still protecting confidentiality requirements?

Objectives

Demonstrate the effectiveness of collaborative multi-provider execution by the realization of a *proof-of-concept* **plan configuration optimizer**

Further requirements:

- feasible and easy integration with existent plans optimizers, for example Spark SQL optimizer [1]
- timely retrieval of near optimal plan configurations (no more than some tenths of a second)

Reference model

Hybrid computation

Our approach supports the integration of *User Defined Functions* with traditional SQL operators. UDFs are modeled as black boxes that correspond to procedural computations constructed using a variety of programming languages and paradigms

Authorization policy

We enforce confidentiality by using the authorization model [2]:

- for each subject S, potentially involved in execution, attributes of schema are split in two visibility levels, plaintext and encrypted, $[P, E] \to S$
- for each operation, a *relation profile* $\left[R^{vp}, R^{ve}, R^{ip}, R^{ie}, R^{\simeq}\right]$, keeps track of implicit (from previously applied operators) and explicit (part of a join) equivalence between attributes during computation
- general formulas are applied to evolve relation profiles according to the specific operation being evaluated
- proper encryption wrapping is applied on-the-fly at attribute granularity level to enforce confidentiality

Operational constraints

We use several cryptographic techniques to carry out computation over ciphertexts without information leaks: Order preserving encryption, Deterministic symmetric encryption, Randomized encryption, Homomorphic encryption

Example of query input plan

As an example of query and authorizations, the data coming from a portable ECG monitor and a fit tracker can be considered:

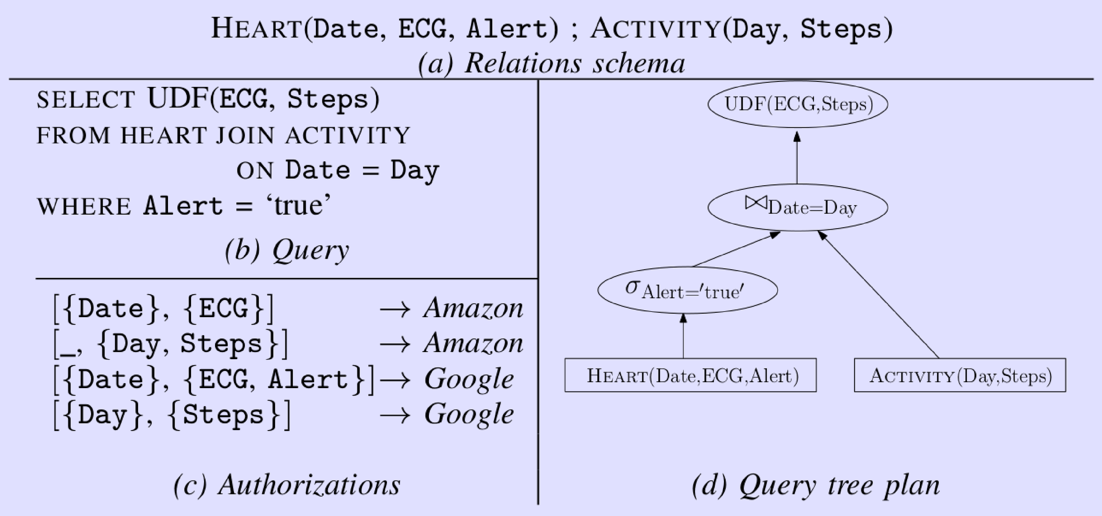


Figure: An example of relations schema, query, authorizations and query tree plan

These data stand as the starting point for our analysis, the objective is to look for the near optimal assignment of the query tree plan

Two-phase optimization

Given a query, we aim at generating a query plan that minimizes the economic cost. Building on a generic optimization chain of the existing SQL query optimizers, we propose a solution based on a **two phase** approach

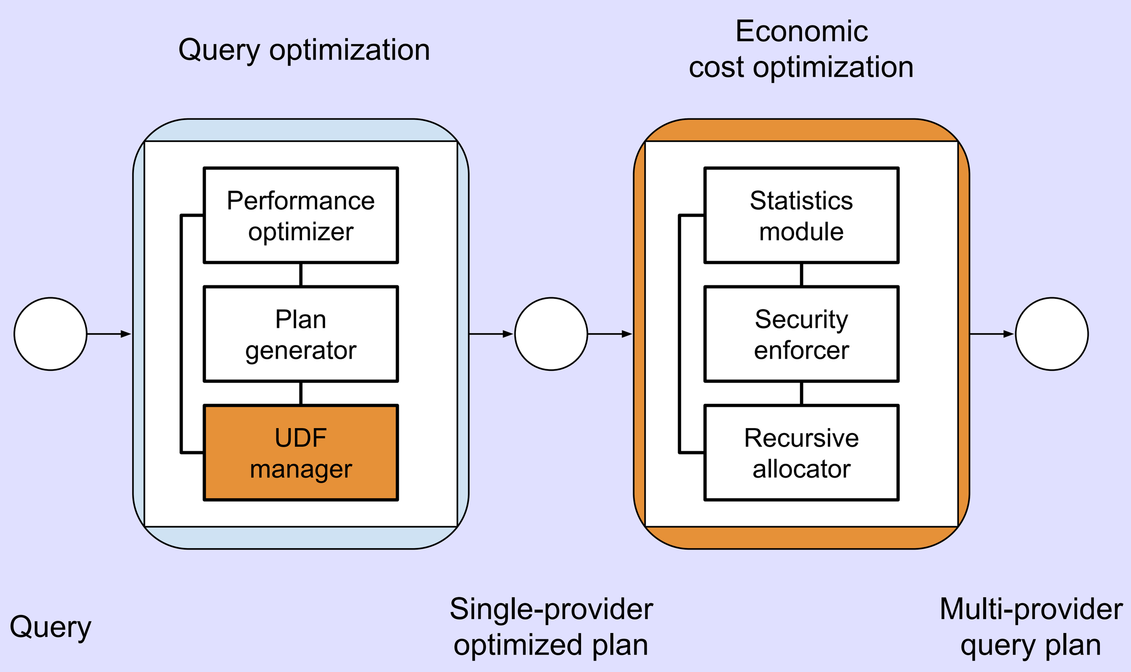
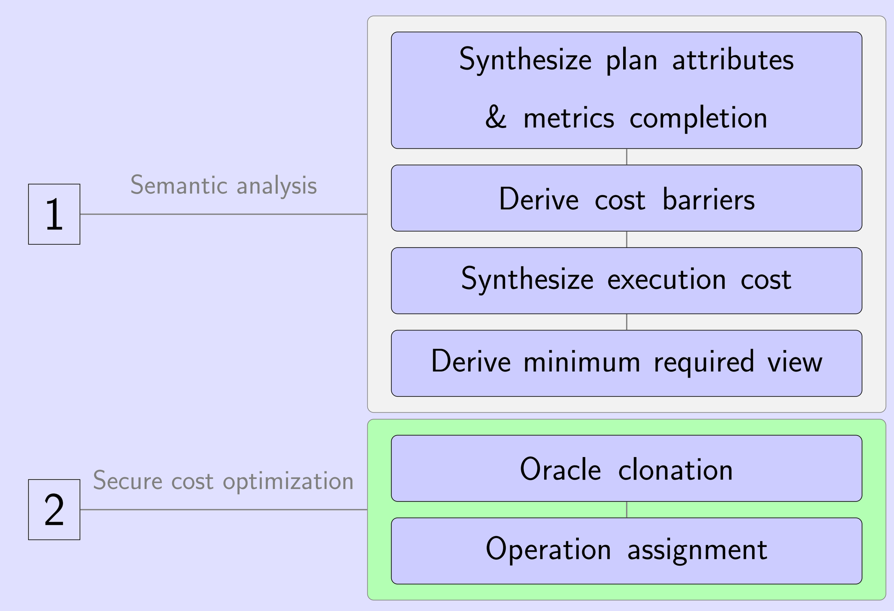


Figure: Two-phase optimization process: *i*) single-provider optimizer (light blue), *ii*) economic cost optimizer (orange)

Cost optimization steps

Before starting executing the cost optimization we carry out some semantic steps: statistics are generated, CPU dominant-cost operations are identified and minimum visibility levels are derived



When the semantic phase is over, the **greedy recursive assignment** phase begins. A clone of the original plan is used to facilitate assignment attempts

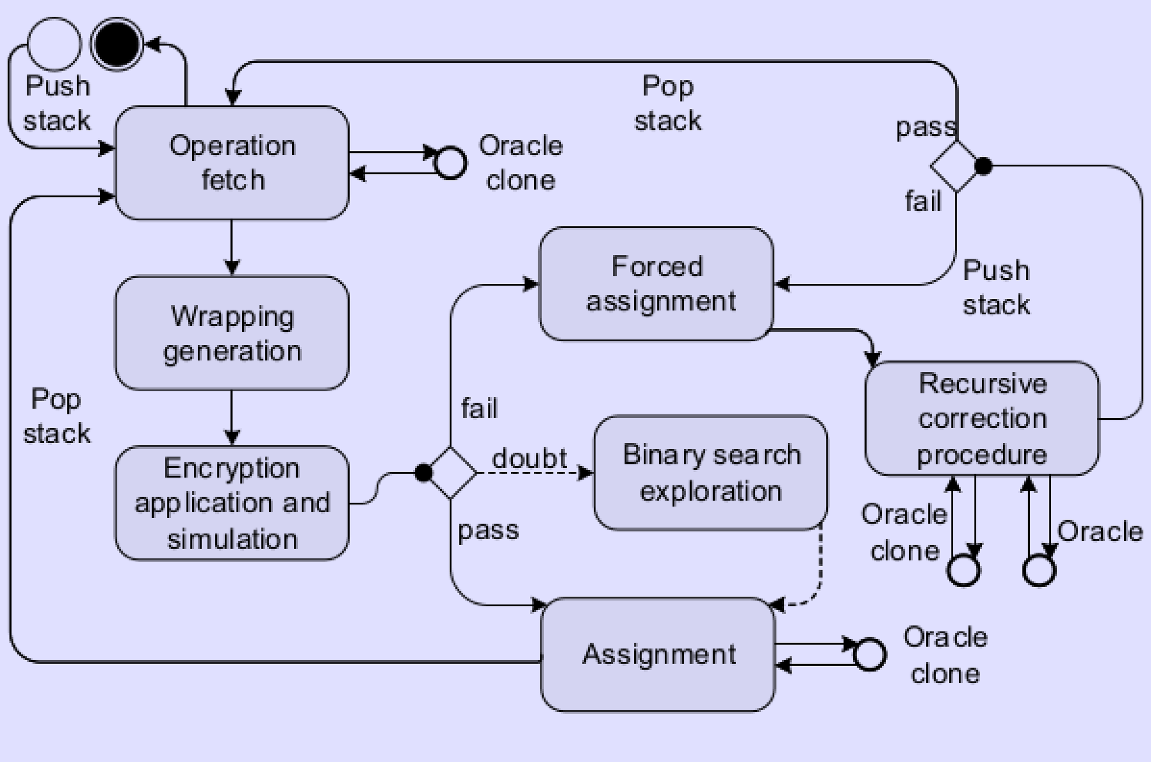
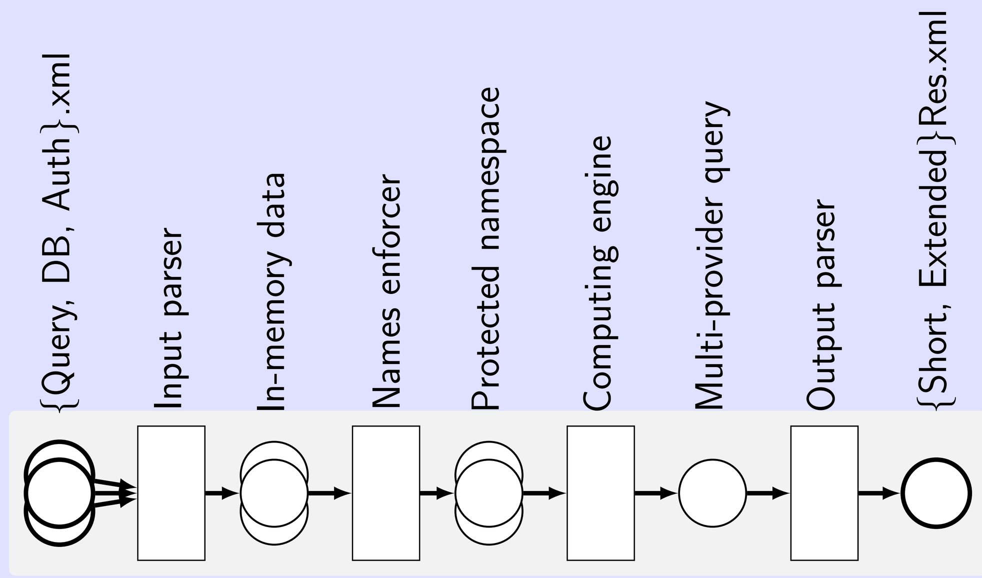


Figure: State machine description of the operations assignment algorithm

Implementation

We implemented the following prototype [3]



The prototype currently supports relational algebra enriched by custom UDF operators, while data format is relational

The *names enforcer* performs pre-processing in order to avoid name clashes

The input DOM parser maps physical query operators to the internal algebraic representation, while the output one acts as a translator, its redefinition permits easy integration in real frameworks

Results and performances

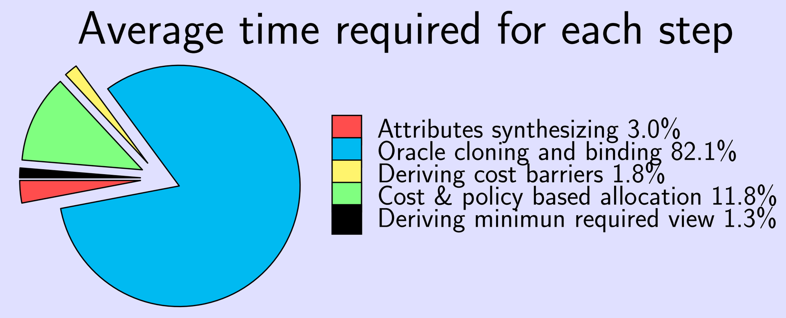
We evaluate our prototype by modeling a **hybrid workload** (SQL + UDF) using three types of **UDF complexity** (γ): linear (L), pseudo-linear (PS) and quadratic (Q)

Table: Cost estimate for each UDF complexity, for each mode

γ	Single-P	Multi-P	Multi-P no_uvr
L	0.041$	0.041$ (0.0%)	0.022$ (53.7%)
PS	0.047$	0.019$ (40.4%)	0.019$ (40.4%)
Q	3.465$	3.465$ (0.0%)	0.497$ (14.3%)

The average cost optimization time is **26.9ms** on an Intel i5 server with 16 GB memory and SSD drive running Ubuntu 18.04 LTS

Average time required for each step



Conclusion

- the implementation and experimental evaluation confirms the efficiency and effectiveness of the proposal, and confirms its compatibility with current query optimization requirements
- the described approach results particularly suited for computations of high complexity

References

Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al.
Spark sql: Relational data processing in spark.
In *Proc. of ACM SIGMOD*, Melbourne, VIC, Australia, 2015.

Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Giovanni Livraga, Stefano Paraboschi, and Pierangela Samarati.
An authorization model for multi provider queries.
PVLDB, 11(3):256–268, 2017.

Query cost optimizer repository.
https://github.com/mosaicrown/query-opt.

Acknowledgments

This work was supported by the EC within the H2020 under grant agreement 825333 (MOSAICrOWN).