Compliant Geo-distributed Data Processing in Action

Kaustubh Beedkar
TU Berlin
kaustubh.beedkar@tu-berlin.de

David Brekardin
TU Berlin
david.brekardin@campus.tu-berlin.de

Jorge-Anulfo Quiané-Ruiz
TU Berlin & DFKI
jorge.quiane@tu-berlin.de

Volker Markl
TU Berlin & DFKI
volker.markl@tu-berlin.de

ABSTRACT

In this paper we present our work on compliant geo-distributed data processing. Our work focuses on the new dimension of dataflow constraints that regulate the movement of data across geographical or institutional borders. For example, European directives may regulate transferring only certain information fields (such as non personal information) or aggregated data. Thus, it is crucial for distributed data processing frameworks to consider compliance with respect to dataflow constraints derived from these regulations. We have developed a compliance-based data processing framework, which (i) allows for the declarative specification of dataflow constraints, (ii) determines if a query can be translated into a compliant distributed query execution plan, and (iii) executes the compliant plan over distributed SQL databases. We demonstrate our framework using a geo-distributed adaptation of the TPC-H benchmark data. Our framework provides an interactive dashboard, which allows users to specify dataflow constraints, and analyze and execute compliant distributed query execution plans.

PVLDB Reference Format:
Kaustubh Beedkar, David Brekardin, Jorge-Anulfo Quiané-Ruiz, and Volker Markl. Compliant Geo-distributed Data Processing in Action. PVLDB, 14(12): 2843 - 2846, 2021.
doi:10.14778/3476311.3476359

1 INTRODUCTION

Today's data analytics applications spread across several databases and IT infrastructures at various international sites. Much research (e.g., [8, 9]) has thus focused on expanding the scope of data processing frameworks to support geo-distributed data processing.

In geo-distributed environments, the location of data is usually predetermined. Therefore, data processing frameworks focus on distributed execution strategies to execute analytical queries. Such strategies typically involve distributing query operators (like join or aggregation) across geo-distributed compute nodes. While research on distributed query processing and optimization has shed lights on various performance aspects (like latency or bandwidth), a new dimension of *cross-border dataflow constraints* [3, 4] has not been considered yet; i.e., dataflow constraints that arise from regulations that prohibit the flow of certain data across geographical borders or from other rules of data protection that may apply to the data being transferred between certain sites. European directives, for example, may regulate transferring only certain information fields (or combinations thereof), such as non-personal information or information not relatable to a person. Likewise, regulations in Asia may also impose restrictions on data transfer. Therefore, we need to consider compliance with respect to legal aspects when distributing operators while generating distributed execution plans, i.e., executing a distributed query must not violate any dataflow constraints. We refer to this kind of distributed query processing as *compliant geo-distributed query processing*.

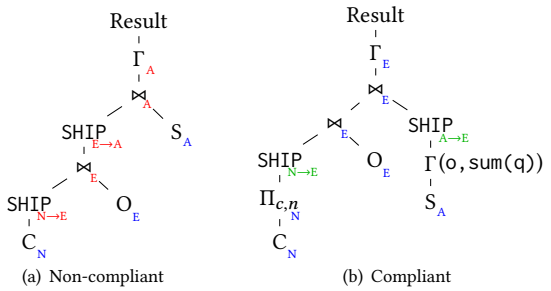
Supporting compliant geo-distributed query processing entails two major research challenges. First, we need an easy (thus declarative) way to specify dataflow constraints. Doing so is not trivial as constraints may pertain to different types of data as well as its processing. For example, restrictions may apply to an entire dataset, parts of it, or even to information derived from it. Second, we have to find efficient ways to process queries in a manner that they are compliant with respect to dataflow constraints. In contrast to cost-based query optimization techniques, which focus solely on performance aspects, we need efficient and effective ways to include compliance aspects in query optimization and processing.

In this paper, we demonstrate a compliance-based query processor based on our earlier publication [2]. The system allows for declarative specification of dataflow constraints. It entails a compliance-based query optimizer, which considers dataflow constraints when generating distributed query execution plans, and a multi-database query executor to execute compliant plans. Our demonstration setup consists of a geo-distributed database comprising five PostgreSql databases (at different locations), and data based on the TPC-H benchmark data. In our demonstration, attendees will be able to specify dataflow constraints using our policy expression language for different locations, and visualize, analyze, and execute distributed execution plans for their queries.

2 DATAFLOW POLICIES & COMPLIANCE

We start with a discussion on dataflow policies and the notion of compliance with respect to distributed data processing that we consider in this paper. As an example, consider a distributed DBMS (DDBMS) comprising three databases D_N, D_E, and D_A located in North America (**N**), Europe (**E**), and Asia (**A**) respectively. D_N stores information about customers, D_E stores information about orders, and D_A stores the supply information. Our example distributed

Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476359



(a) Non-compliant (b) Compliant

Figure 1: Example execution plans for Q_{ex}.

database has the following schema and data distribution

Customer	(**c**ustkey, **n**ame, **a**cctbal, **m**ktseg, **r**egion)	**N**
Orders	(**c**ustkey, **o**rdkey, **t**otprice)	**E**
Supply	(**o**rdkey, **q**uantity, **e**xtprice)	**A**

Dataflow Policies. A *cross-border dataflow policy* describes the restrictions on the transfer of data across organizational and/or geographical borders. Generally speaking, a dataflow policy specifies *which* information as well as *how* and to *where* this information can be legally transferred. For example, based on recent studies on data movement regulations [1, 3, 4], consider dataflow policies $\mathscr{P}_N$, $\mathscr{P}_E$, and $\mathscr{P}_A$ that applies to data from North America, Europe, and Asia respectively:

$\mathscr{P}_N$ Customer data can be shipped outside only after suppressing account balance information.

$\mathscr{P}_E$ Only aggregated Orders data can be shipped to Asia and an order's price cannot be shipped to North America.

$\mathscr{P}_A$ Only aggregated Supply data for orders' quantity and extended price from Asia can be shipped to Europe.

Compliant Geo-distributed Data Processing. In geo-distributed environments, typical strategies to process a query [5, 7–9] — that involves transferring intermediate results between sites— may not comply with data movement regulations. To illustrate, consider a query Q_{ex}

```
SELECT C.name, SUM(O.totprice), SUM(S.quantity)
FROM Customer AS C, Orders AS O, Supply AS S
WHERE C.custkey=O.custkey AND O.ordkey=S.ordkey
GROUP BY C.name
```

Figure 4 shows two query execution plans (QEP) for Q_{ex}. Here, the SHIP operator describes the point where intermediate results are communicated between two sites and Γ denotes the aggregation operator. Assume now the QEP in Figure 1(a) is more efficient than the QEP in Figure 1(b). In this case, a traditional DDBMS, which uses cost-based query optimization strategies, most likely will output the QEP in Figure 1(a). However, this plan is non-compliant: its SHIP operators violate dataflow policies $\mathscr{P}_N$ ($\text{SHIP}_{N\to E}$ ships Customer table without suppressing the account balance) and $\mathscr{P}_E$ ($\text{SHIP}_{E\to A}$ ships non-aggregated Order information to Asia). In contrast, the QEP in Figure1(b) is compliant: it performs both join operations in Europe, $\Pi_{c,n}$ suppresses the account balance information of Customers and $\Gamma(\texttt{o,sum(q)})$ suppresses (via aggregation) the orders' quantity.

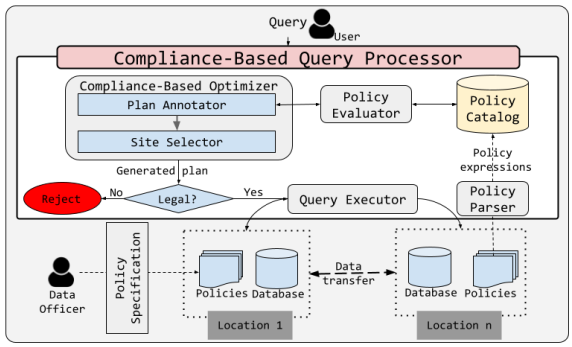


Figure 2: Compliant geo-distributed data processing

3 COMPLIANCE-BASED QUERY PROCESSOR

We now describe our compliance-based query processing framework, which (i) allows declarative specification of dataflow constraints, (ii) determines if a query is legal w.r.t. the imposed constraints, and (iii) translates a legal query into a compliant QEP. Figure 2 illustrates our overall approach. In what follows, we first give an overview of our system and then discuss its core components: policy specification, query optimizer, and a multi-database query executor.

3.1 System Overview

We consider a distributed SQL database composed of autonomous geo-distributed databases. Each database is tied to a location[1]. For each database, the framework allows a data officer to reflect her dataflow policies using *policy expressions* (Section 3.2), which are stored in a policy catalog. Note that this policy specification process is an offline process. At querying time, the compliance-based query optimizer (Section 3.3) uses this policy catalog (via the policy evaluator), when enumerating plans, to validate if they are compliant with their input dataflow policies. The optimizer uses this validation mechanism to know when a QEP is violating an existing dataflow policy. Only when the resulting QEP is compliant, it proceeds with the query execution. The query executor then executes the query over the distributed databases. Our current implementation supports such a multi-database query execution over PostgreSql databases.

3.2 Policy Specification

A crucial aspect in adhering to dataflow policies is to mask data in a way that renders it suitable to be transferred across borders. In this work, we focus on policies that can be adhered to by masking via relational operations (e.g., project, aggregate, or filter) such that the resulting compliant QEP retains the query semantics, i.e., the output of the query should be the same if there were no dataflow policies. For instance, a projection operator can mask certain columns by projecting them out before the (intermediate data) is shipped to another location, and when the masked columns are not required by the query later. In this context, we propose *policy expressions* as

[1]Here location can be geographical, institutional, or the database instance itself.

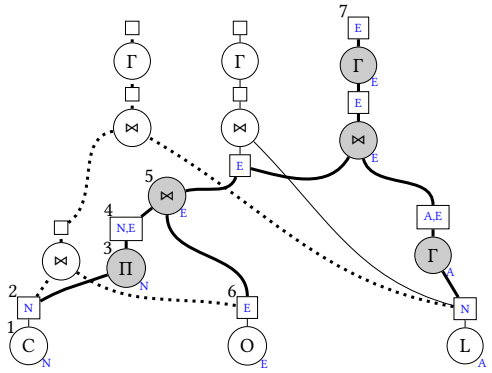


Figure 3: Simplified search space for Q_{ex}.

a simple and intuitive way to specify dataflow policies. We define two kinds of policy expressions: basic and aggregate expressions.

Basic Expressions. Basic expressions allow specifying shipping of certain rows and columns of a table to another location and have the following syntax:

```
ship attribute list from table to location list
where condition list
```

This expression specifies cells, i.e., rows and columns, of a table to be shipped without affecting the query semantics.[2] The specified cells from the table in the **from** clause (i) belong to both columns in the **ship** clause and tuples that satisfy the predicates in the **where** clause, and (ii) can be shipped to locations in the **to** clause. Intuitively, if a subquery accesses only the specified cells, then its output can be shipped to locations specified in the expression.

Consider policy $\mathscr{P}_N$ from Section 2, which does not allow for shipping the account balance information of customers outside North America. Suppose the policy also allowed for shipping customer's mktsegment and region information to Europe for commercial customers. We can use the following policy expressions:

```
ship custkey, name from Customer C to Asia, Europe

ship mktseg, region from Customer C to Europe
where mktseg='commercial'
```

Aggregate Expressions. Although basic expressions are sufficient to express a large variety of dataflow policies, there are policies that allow for shipping of aggregate information only. For these cases, we have aggregate expressions that allow us to specify aggregations over columns. Similar to basic expressions, aggregate expressions do not affect the query semantics. The syntax of an aggregate expression is:

```
ship attribute list as aggregates aggregate types
from table to location list where condition list
group by attribute list
```

In the above syntax, the list of attributes in the **ship** clause specifies cells of columns that should be aggregated before being shipped to locations in the location list. The **as aggregate** clause specifies aggregation functions that should be used to aggregate specified cells. As before, the specified cells must belong to columns in the attribute list for the tuples that satisfy the predicate in its **where** clause. Lastly, the **group by** clause specifies lists of grouping attributes for which the specified cells can be grouped by zero, one, or more attributes from its attribute list.

Consider again the Customer table from Section 2 and assume that account balance information can be shipped only after aggregating. A possible expression is:

```
ship acctbal as aggregates sum, avg from Customer C
to * group by mktseg, region
```

[2]For exposition, we restrict to expressions over a single table. This is not a limitation: one can specify a policy expression over more than one base table. In this case, the condition list in the **where** clause of the expression must contain the join predicate.

3.3 Compliance-based Optimizer

We now describe our compliance-based optimizer. The goal of our optimizer is to determine if a query is legal and to automatically generate and a compliant plan.

We follow a two-phase optimization process that comprises plan annotation and site selection. The plan annotator receives a logical plan as input and outputs an annotated QEP. An *annotated QEP* is an optimized logical plan in which each plan operator is annotated with a set of compliant sites (i.e., sites where the execution of the operator will not violate any dataflow constraint). The site selector then uses dynamic programming to find the optimal placement of query plan operators taking data shipping cost into account.

More specifically, we adapt the Volcano optimizer generator [6] to generate the plan annotator. Our adaptations allow us to produce an annotated plan by enumerating the plan space by applying algebraic equivalence rules in a top-down fashion and filter compliant ones by applying our annotation rules in a bottom-up fashion. To do so, we treat geo-locations associated with base tables as "interesting properties" and propagate these properties bottom-up via annotation rules. Our annotation rules are based on the structure of the subplans and make use of a lightweight mechanism (the policy evaluator; Figure 2) to evaluate dataflow policies. Our policy evaluator allows for easy integration of policy expressions into the annotation process. In particular, during plan enumeration, it determines to which cross-borders sites the output of operators can be shipped.

The annotation process for our running example is illustrated in Figure 3. Here the plan with the dotted lines shows the initial logical plan. The plan with thick solid lines shows the annotated plan that the annotator outputs. The letters in the square boxes denote sites to which the output of an operator can be shipped to and letters below each operator denote sites where each plan operator can be executed. For example, the project operator (node 3) must be executed in North America but its output can be shipped to North America and Europe. It is easy to see that the plan with thick solid lines translates to the compliant plan shown in Figure 1(b). For a more detailed description of our optimizer, we refer readers to [2], which also gives a more formal treatment and proof of correctness.

3.4 Query Executor

Lastly, the query executor receives a compliant (logical) plan in which a global property describes where the processing of each plan operator must happen. The executor then orchestrates the multi-site query execution by translating the compliant plan into a sequence of DDL statements corresponding to underlying database systems.

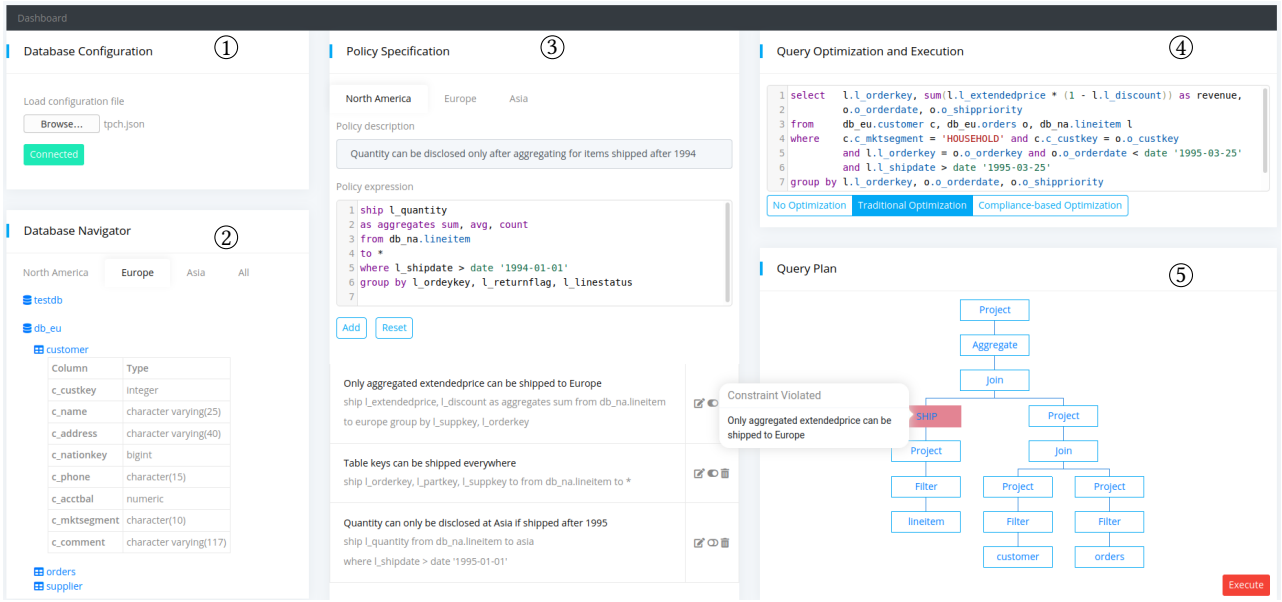


Figure 4: User interface for demonstration

4 DEMONSTRATION

We demonstrate our framework using a geo-distributed adaptation of the TPC-H benchmark data. Our framework supports compliant geo-distributed data processing over data stored in disparate PostgreSql databases, which are located at five locations. We provide an interactive dashboard, as shown in Figure 4 to demonstrate various aspects of our system. The following scenarios will be available to the attendees.

Plug-and-Play Database Connection. The attendees will be able to setup our framework atop existing PostgreSql databases, via the Database Configuration pane ①. Attendees can select one of the available configurations of our distributed TPC-H database. Each differing in how TPC-H tables are distributed among different locations. Attendees can also specify details of their own PostgreSql database and plug-in to our framework with no extra effort.

Geo-distributed Data Exploration. After the framework successfully connects to the databases, the Database Navigator pane ② will allow attendees to explore the schema of the geo-distributed data in a unified way. Attendees can navigate through individual databases, view the table distribution, and schema of the databases.

Policy Specification. During the demonstration, an attendee will take the role of a data officer and reflect her dataflow policies. The Policy Specification pane ③ provides an interface to specify data movement policies using our policy expressions. The attendee can add new constraints and also update, delete, or disable already added constraints with respect to one or more locations.

Geo-distributed Data Processing. Once the constraints have been added, attendees can submit ad-hoc SQL queries. The Query Optimization and Execution pane ④ provides a query interface, using which the attendee will first generate a distributed execution plan either by invoking our compliance-based optimizer, or by invoking a traditional cost-based optimizer, or she may generate an unoptimized plan. The Query Plan pane ⑤ will show the generated plan and also allow the attendee to interact with query plan operators to view details such as their parameters. If the resulting plan is not compliant, then the ship operators will be marked with red color, otherwise, they are marked using green color (not shown here). For non-compliant plans, the Query Plan pane will show an inactive red execute button. Attendees can click on the ship operators to see which policies were violated by the query. Lastly, attendees will be able to execute compliant plans, generated by our compliance-based optimizer, in a multi-site setting.

ACKNOWLEDGMENTS

This work was funded by the German Ministry for Education and Research as BIFOLD – “Berlin Institute for the Foundations of Learning and Data” (01IS18025A and 01IS18037A).

REFERENCES

[1] 2019. Regional Privacy Frameworks and Cross-Border Data Flows. https://www.gsma.com/publicpolicy/resources/regional-privacy-frameworks-and-cross-border-data-flows.
[2] Kaustubh Beedkar, Jorge-Arnulfo Quiané-Ruiz, and Volker Markl. 2021. Compliant Geo-Distributed Query Processing. In *SIGMOD*. 181–193.
[3] Francesca Casalini and Javier López González. 2019. Trade and Cross-Border Data Flows. 220 (2019). https://doi.org/https://doi.org/10.1787/b2023a47-en
[4] Nigel Cory. 2017. *Cross-Border Data Flows: Where Are the Barriers, and What Do They Cost?* http://www2.itif.org/2017-cross-border-data-flows.pdf
[5] A. Deshpande and J. M. Hellerstein. 2002. Decoupled query optimization for federated database systems. In *ICDE*. 716–727.
[6] Goetz Graefe and William J. McKenna. 1993. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *ICDE*. 209–218.
[7] Donald Kossmann. 2000. The State of the Art in Distributed Query Processing. *ACM Comput. Surv.* 32, 4 (2000), 422–469.
[8] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low Latency Geo-distributed Data Analytics. In *SIGCOMM*. 421–434.
[9] Ashish Vulimiri, Carlo Curino, Brighten Godfrey, Konstantinos Karanasos, and George Varghese. 2015. WANalytics: Analytics for a Geo-Distributed Data-Intensive World. In *CIDR*.