A Polystore Querying System applied to heterogeneous and horizontally distributed data

Lea El Ahdab*, Olivier Teste*, Imen Megdiche*, and Andre Peninou*

*Université de Toulouse, IRIT, Toulouse, France
{lea.el-ahdab,olivier.teste,imen.megdiche,andre.peninou}@irit.fr

Abstract. Data storage in various systems such as SQL and NoSQL leads to important problems when trying to unify data querying. Multiple storage systems conduct to heterogeneous data structures and to multiple query languages. In the context of horizontally and disjointed distributed data, this paper proposes a system that allows the user to natively query a polystore system without taking care of data distribution and heterogeneity. Our approach relies on two mechanisms: (i) mapping dictionaries to define the navigation between systems, (ii) operator rewriting mechanisms from native query operators (selection, projection, aggregation and join) to execute queries on any polystore system. Using a dataset from TPC-H benchmark and a horizontally distributed between document and relational database management system, we conduct experiments showing that the rewriting process has a minimum impact when compared to executing queries in both systems.

Keywords: Polystore · Heterogeneity · Data distribution.

1 Introduction

Nowadays, data is more likely to be found distributed in classical (SQL) or multiple heterogeneous and flexible data sources (NoSQL), which forms polystores [2]. It complexifies querying in multiple languages based on non-standardized data modeling paradigms and data querying operators. New solutions are based on new languages [7], operators [10] [8], models [6], and sometimes flexible schemas [4] [3], which depend on data manipulation. This paper deals with horizontal data distribution in which one entity class is stored in different datastores (relational and document-oriented). We introduce a solution for querying polystore systems, based on automatic rewriting and decomposition of queries, facilitating access to horizontally distributed and heterogeneous data using a mapping dictionary able: i) to link each attribute of any dataset to the corresponding attributes in other datasets and, ii), to integrate possible heterogeneity of data inside any dataset. Working on a TPC-H dataset, we experiment our solution with a query rewriting process without impacting the initial query execution time on relational databases and document-oriented datastores. The remainder of the paper is structured as follows. In section 2, we discuss existing solutions and present their limits. Section 3 defines our polystore data model. Section 4

presents and illustrates the proposed rewriting process and the mapping dictionary with data distribution. Section 5 shows our solution results on real data. Finally, in section 6 we give some perspectives about future work.

2 Related Work

With the complexity of data storage systems in polystores (distribution and heterogeneity), query and accessibility should stay as simple as it is in a mono system type store. Some works focus on inferring schemas to access data: graph representation [4] [5] or a u-schema model [3] showing structural variations. Existing works mainly focus on vertical distribution where each entity class is found in one database. Some systems [3] [8] introduce an external function to manipulate several entities for binary operators. The join operator is not always executed inside DBMS which requires to have an external algorithm joining the sub results [9] [8]. However, horizontal data distribution is possible, where every entity class is divided inside multiple databases: user query gets complex and should be expressed by taking into account data location, query formulation according to polystore systems languages. Another aspect is data heterogeneity: semantic [2] [9] or syntactic [4] issues. They use synonyms in mapping solutions to build their queries [2] [9]. Surprisingly, they do not deal with structural heterogeneity which is induced by the schema-less principle of NoSQL stores. A specific query is translated and parsed into languages of the considered datastores [2] [9]. To support our comparison with existing works, table 1 illustrates the differences we can find between our works and others working on query rewriting and mapping. It shows their position about data distribution (H: horizontal, V: vertical), the supported systems (Relational, Document, Column, Graphs) and the query expression with operators. It also provides the query of relational and document-oriented systems in their native languages using algebraic equivalences and presents to the user results in their native form without transforming data. Our solution consists in the rewriting of the combination of SPAJ operators (selection, projection, aggregation and join).

3 Algebraic definition of Polystores for horizontally distributed data

In SQL approaches, data is represented according to the relational model [1], where data is structured according to relation schemas. NoSQL (documents) approaches are "schema-less" - each record has its own structure that may be different from those of other records in the same dataset. A **polystore** system is defined as $PL = \{DB_1 , ..., DB_B\}$ where each **database** is $DB_i = \{ DS_1 , ..., DS_{S_i} \}$. $\forall j \in [1 ... S_i]$, DS_j is a dataset. Our model gives a universal representation of these different databases. Each **dataset**, DS_j, is defined by an extension and an intention $DS_j = (Int_j , Ext_j)$. An **extension** is a set of instances $i_k = (\chi_k, v_k)$. χ_k is its *key*, internal identifier in database systems, and v_k is the *instance value* which can be atomic or recursively an instance

Table 1. A comparison of existing solutions on polystores

Authors	Data Distrib.	R	D	C	G	Heterogeneity	Query	σ	π	γ	$\bowtie$
El Ahdab et al	H	✓	✓			Structural Semantic Syntactic	SQL MongoDB	✓	✓	✓	✓
Barret et al [4]	V	✓	✓		✓	Syntactic	SparkQL				
Candel et al [3]	V	✓	✓		✓		SQL	✓	✓		
Ben Hamadou et al [5]	V	✓	✓	✓		Structural Semantic	SQL MongoDB	✓	✓	✓	✓
Hai et al [9]	V	✓	✓		✓	Semantic	SQL JSONiq	✓	✓	✓	✓
Duggan et al [2]	V	✓	✓	✓		Semantic	Declarative	✓	✓		✓
Curino et al [10]	V	✓				Structural	SQL	✓	✓		

value or an array of values. The **intention** inferred from the extension is the set of all absolute paths deduced from all instance structures existing in the extension $Int = \bigcup_{k=1}^{N_j} S_k$. We focus on polystores where $\forall i_1 \in [1...B]$, $DS_{j_1} \in DB_{i_1}$, $\exists i_2 \in [1...B]$ such as $DS_{j_2} \in DB_{i_2}$ and DS_{j_1}, DS_{j_2} contain different instances of the same class of an entity. A horizontal distribution is **strict** when each attribute of a dataset has at least one equivalent designation in all equivalent datasets. A data distribution is *disjointed* when $\forall i_{k_1} \in Ext_{j_1}, i_{k_2} \in Ext_{j_2}, j1 \neq j2 \mid v_{k_1} = v_{k_2}$ where v_{k_1} and v_{k_2} are values corresponding to the same entity in the real world. The **mapping dictionary** map_{DS_j} matches each path of a dataset to all its corresponding paths (including itself) in all equivalent datasets dealing with *structural*, *syntactic* and *semantic* heterogeneity. Due to space limitation, we do not detail in this paper how the mapping dictionaries are built; they are maintained with the definition and using data alignment and schema-matching algorithms [4]. For example, a path A in DS_i is mapped with every corresponding paths in the equivalent dataset DS_j as: $\{(A, DS_j), (X.A, DS_j), (A'.DS_j)\}$.

4 Rewriting process definition

We introduce a closed set of operators to formalize a universal algebra: $K = \{\sigma\,,\,\pi\,,\,\gamma\,,\,\bowtie\}$ where σ is a selection operator (restriction), π is a projection operator, γ is an aggregate operator and $\bowtie$ is a binary operator used to join two datasets. Their combination formulates a query $Q = q_1 \circ ... \circ q_r$ where $\forall k \in [1 \,...\, r], q_k$ is a simple operator or a composition of operators as a sub-query itself. Each q_k of Q is rewritten according to the mapping dictionaries of the queried datasets $map_{DS_{in}}$. A list of mappings for one field f_i is inferred from the set identified in its respective dictionary.

Selection. $\sigma_P(DS_{in})$ is rewritten as $\sigma_{P_{new}}(DS_j)$ where DS_j is a targeted dataset during query rewriting process and the rewriting of P is P_{new} =

$\wedge(\vee(\vee\, p_{k_l}\ \omega_k\ v_k\,))$ where p_{k_l} are the paths obtained from the rewriting dictionary associated to DS_j and that corresponds to f_i.

Projection. $\pi_E(DS_{in})$ is rewritten as $\pi_{E_{new}}(DS_j)$ where DS_j is a targeted dataset and E_{new} is the rewriting of $E = e_1, ..., e_n$. If $e_i = f_i$ (projection): f_i is replaced by the combination of its corresponding absolute paths according to the mapping dictionary: $p_{k_1}|\ldots|p_{k_m} \forall p_{k_l}$ for DS_j. The "|" operator leads to the projection of the existing path p_{k_l} in any instance value of Ext_{DS_j}. If $e_i = f_i' : f_i$ (projection and renaming): f_i is replaced by $f_i' : p_{k_1}|\ldots|p_{k_m}$ for all DS identified.

Aggregation. ${}_G\gamma_F(DS_{in})$ is rewritten to a dataset DS_j as ${}_G\gamma_F(\pi_{Enew}(DS_j))$ where E_{new} is a projection rewriting of fields of $G \cup \{f_i\}$. The projection on f_i of the function F is rewritten to $f_i : p_{k_1}|\ldots|p_{k_m} \forall p_{k_l} \in \Delta_{f_i}^{DS_j}$. The same process is applied to all fields of G.

Join. $DS_{in1} \bowtie_J DS_{in2}$ is rewritten for a database DB_j using the corresponding datasets of DS_{in1} and DS_{in2} in DB_j as: $DS_{j_{in1}} \bowtie_{J_{new}} DS_{j_{in2}}$. J_{new} corresponds to the join condition containing the mapped fields.

The user queries one dataset in one language and the query is translated in its algebraic form. Query rewriting rules are used to produce B queries, one for each DB_i of PL. Rewritten queries are then translated into their specific language (SQL or MongoDB) before being executed on DB_i. The sub-results are presented to the user in their original form (using JSON notations). In some complex queries, they may represent only intermediate results and may need more computation to give the target result; in case of aggregation using sum function, some cases may require an additional aggregation to sum intermediate results.

5 Experiments

TPC-H Benchmark. Considering TPC-H data (https://www.tpc.org/) and queries, we have stored data in one SQL database (MySQL) and one Document oriented database (MongoDB). Tables and collections were created in each respective systems. We have considered two volumes of data v1 as 1 Mo (3600 tuples) and v2 as 10 Mo (30000 tuples).

Query rewriting evaluation TPC-H queries are classified according to the number of queried datasets and to their operators composition. Almost half of them are an association of selection (σ), projection (π), aggregation (γ) and joins ($\bowtie$). Our evaluation focuses on (1) analyzing query execution time over an equal data distribution inside both system and when this distribution is unbalanced, and (2) on the impact of operations on the query execution time in the same context of data distribution. We considered a condition of fragmentation on *nation name* and which respects a disjointed repartition of 50% of instances in

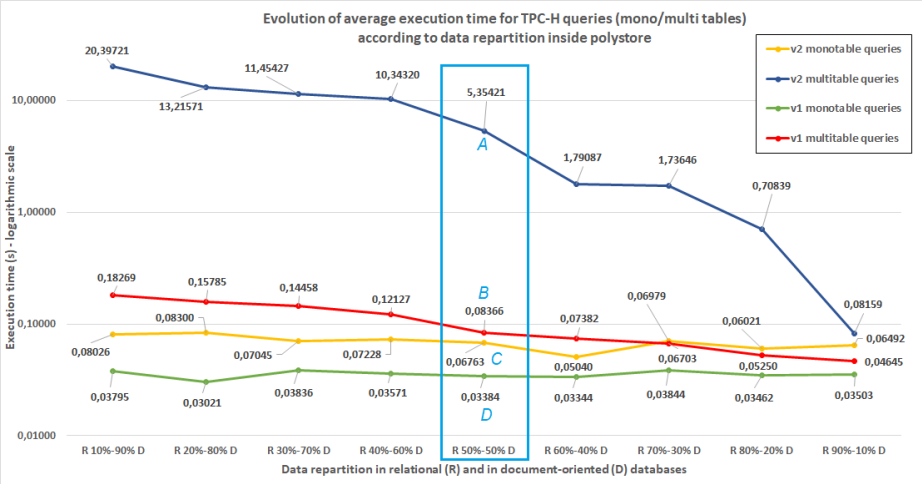

Fig. 1. Evolution of average execution time for TPC-H queries according to data distribution inside polystore with different data volumes v1 (1Mo) and v2 (10Mo)

the relational database and 50% in the document-oriented one. We have evolved this distribution to consider other situations (10%-90%, 20%-80%...).

As illustrated in figure 1, the relational system shows a lower execution time than the document system. Focusing on the 50%-50% distribution, SPA operations have no impact on execution time but the join operation presents a higher difference between systems: execution time is 80% times higher for multitable queries than monotable queries (value A, value C of figure 1). When data is distributed 90% in documents, query rewriting time is maximize in comparison of 90% of data distribution inside relations. Since each query is executed in each database, it results in a set of separate pieces of data, with possible different structures presented to the user (tuples and documents).

6 Conclusion

In this paper we focus on polystore systems with relational and document-oriented datasets, where data is distributed horizontally. A mapping dictionary represents links between fields and their correspondences in every data source and in their heterogeneous forms. A universal query algebra composed of SPAJ operators is defined for querying both considered systems supporting query rewriting rules and bringing transparency for users. Data remain in native form and only dynamic rewriting of queries and the mapping dictionary are impacted by eventually new data structures. Experiments on a TPC-H dataset show the effectiveness of the proposed solution without significantly impacting the query execution time on top of relational databases (MySQL) and document-oriented databases (MongoDB). In the future, we will focus on the extension of the exist-

ing algebra to other systems (column, graph). Another direction is to consider operators more specific to storage systems in order to find their rewriting forms.

Acknowledgements This work was supported by the French Gov. through the Territoire d'Innovation program, an action of the *Grand Plan d'Investissement* backed by France 2030, Toulouse Métropole and the GIS neOCampus.

References

1. CODD, Edgar F. Further normalization of the data base relational model. Data base systems, 1972, vol. 6, p. 33-64.
2. DUGGAN, Jennie, ELMORE, Aaron J., STONEBRAKER, Michael, et al. The bigdawg polystore system. ACM Sigmod Record, 2015, vol. 44, no 2, p. 11-16.
3. CANDEL, Carlos J. Fernández, RUIZ, Diego Sevilla, et GARCÍA-MOLINA, Jesús J. A unified metamodel for NoSQL and relational databases. Information Systems, 2022, vol. 104, p. 101898.
4. BARRET, Nelly, MANOLESCU, Ioana, et UPADHYAY, Prajna. Abstra: Toward Generic Abstractions for Data of Any Model. In: Proceedings of the 31st ACM International Conference on Information & Knowledge Management. 2022. p. 4803-4807.
5. BEN HAMADOU, Hamdi, GALLINUCCI, Enrico, et GOLFARELLI, Matteo. Answering GPSJ queries in a polystore: A dataspace-based approach. In : Conceptual Modeling: 38th International Conference, ER 2019, Salvador, Brazil, November 4–7, 2019, Proceedings 38. Springer International Publishing, 2019. p. 189-203.
6. DANIEL, Gwendal, GÓMEZ, Abel, et CABOT, Jordi. UMLto [No] SQL: mapping conceptual schemas to heterogeneous datastores. In : 2019 13th International Conference on Research Challenges in Information Science (RCIS). IEEE, 2019. p. 1-13.
7. MISARGOPOULOS, Antonis, PAPAVASSILIOU, George, GIZELIS, Christos A., et al. TYPHON: Hybrid Data Lakes for Real-Time Big Data Analytics–An Evaluation Framework in the Telecom Industry. In : Artificial Intelligence Applications and Innovations. AIAI 2021 IFIP WG 12.5 International Workshops: 5G-PINE 2021, AI-BIO 2021, DAAI 2021, DARE 2021, EEAI 2021, and MHDW 2021, Hersonissos, Crete, Greece, June 25–27, 2021, Proceedings. Cham : Springer International Publishing, 2021. p. 128-137.
8. KOLEV, Boyan, VALDURIEZ, Patrick, BONDIOMBOUY, Carlyna, et al. CloudMdsQL: querying heterogeneous cloud datastores with a common language. Distributed and parallel databases, 2016, vol. 34, p. 463-503.
9. HAI, Rihan, QUIX, Christoph, et ZHOU, Chen. Query rewriting for heterogeneous data lakes. In: Advances in Databases and Information Systems: 22nd European Conference, ADBIS 2018, Budapest, Hungary, September 2–5, 2018, Proceedings 22. Springer International Publishing, 2018. p. 35-49.
10. CURINO, Carlo A., MOON, Hyun Jin, DEUTSCH, Alin, et al. Update rewriting and integrity constraint maintenance in a schema evolution support system: PRISM++. Proceedings of the VLDB Endowment, 2010, vol. 4, no 2, p. 117-128.