Unified views for querying heterogeneous multi-model polystores

Léa El Ahdab, Olivier Teste, Imen Megdiche, André Péninou

Unified views for querying heterogeneous multi-model polystores

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 SQL and NoSQL systems brings complexity to data querying when entities are fragmented because data is not always stored in the same system, plus heterogeneous structures can appear for entities. A unique query language is not sufficient to address data distribution and heterogeneity. Considering vertically distributed data, this work implements a framework capable of rewriting a user query addressed over a unified view to access all data and provide results with transparency. Our framework works with a conceptual model producing unified views to guarantee polystore querying without having to know data distribution nor data heterogeneity. It complements the initial query with intermediate operations. It is applied on an e-commerce scenario (UniBench benchmark) distributed vertically between relational and document-oriented databases. Performance results and the low impact of query rewriting process are illustrated in this work.

Keywords: Polystore · Heterogeneity · Data distribution.

1 Introduction

Various storage systems have emerged and constitute polystore systems that federate SQL and NoSQL data stores. Querying a polystore without a unique model is complicated due to databases diversity and data distribution. Solutions have appeared focusing on vertical data distribution [1] [2] [3]. The distribution of one entity class over several databases is not considered in such works. In this article, we introduce a framework for querying a multi-model polystore system with vertically distributed entities. The framework provides unified logical views of the polystore in relational or document model. The user queries over one of these logical views which serves as a pivot representation for translating user queries into the different paradigms of the multi-model polystore, guaranteeing transparency of data distribution and to data heterogeneity. In section 2 of this paper, we explain our scope with a motivating example based on an e-commerce scenario. Section 3 discusses existing solutions and their limits. Section 4 defines query construction process and section 5 shows results of our experiments on real data. In the last section, we conclude on this work and we give some perspectives about the future ones.

2 Motivating example

Based on an E-Commerce scenario from the multi-model benchmark UniBench [11], we consider four entity classes distributed in two family systems: two relational databases (DB1 and DB2) and one document-oriented database (DB3) . DB1 contains Customers entity, DB3 contains Reviews Entity. Entities *Product* and *Orders* are vertically fragmented into DB2 and DB3 and their ids product_id and order_id are the fragmenting key. The relationship Order_Line is included in Orders of DB3 (fig. 1). The consideration of document-oriented databases brings possible data heterogeneity. In this example, entity classes Orders and Reviews contains different structures: different structures for nested values for order_line, optional values for feedback.

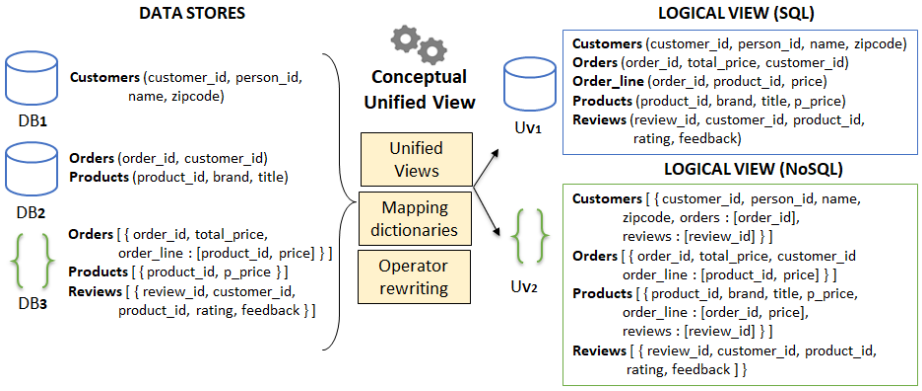

Fig. 1. Our multi-model framework based on logical views and one unified internal conceptual view ensuring data location and data model equivalences

Let us consider the query: analysis of order prices and brand per customer within the best rating products (>=5). Such query requires the user to query SQL tables and documents collections to retrieve and join both Customers, Reviews and build additional joins to retrieve Products and Orders/order_lines. Our approach is based on unified logical views that present all the data either in relational (UV_1 in fig 1) or document-oriented (UV_2 in fig 1) form. These logical views are used to hide data distribution in the polystore and their various modeling paradigms. The user builds a query against one of these logical views. Our system works to generate executable sub-queries on the different databases which are connected using joins over a specific property of the fragmented entities (fig 2). This process potentially induces data transfers. It works on the algebraic tree of the query and transforms it to insert necessary joins to resolve data distribution and "rebuild" fragmented entities when necessary. A final step transforms the algebraic tree to insert data transfers and transformations. The final result is presented in the form of the unified logical view used for querying.

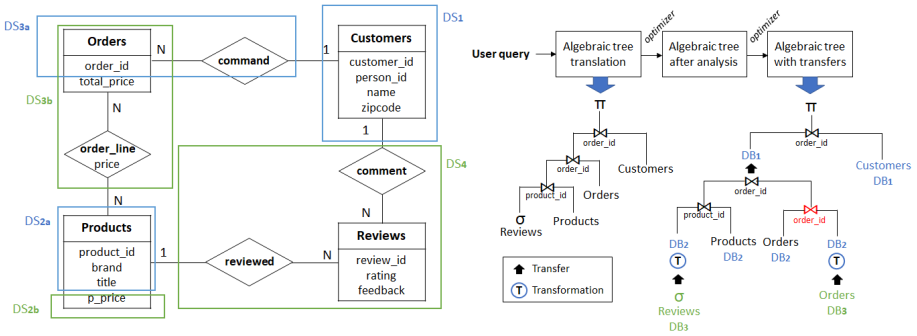

Fig. 2. Application of our framework on the presented use case's query using the associated conceptual model

3 Related Work

Combining SQL and NoSQL systems in one infrastructure, called polystore, brings the notions of multi-store, heterogeneity and data distribution. With vertical distribution where one entity class is found in one datastore of the polystore, unary operators are executed on one system and the binary operator *join* is executed outside DBMS with an external function [1] [5] [12]. HydRa [10], a framework, mentions entity fragmentation but do not explain how to considerate it for querying purposes. Inferring schemas is proposed to unify querying the polystore's data. It can be a graph representation [6] [7] or a u-schema model [5] illustrating structural variations. It brings the issues of query language(s) to access data and the modification of data storage each time data is manipulated. Changing data representation impacts users and modifies the initial paradigm presented to them. A unifying model does not work on data heterogeneity. Some works focus on semantic heterogeneity [3] [8] or syntactic [6] issues provided by the multi-storage environment. Structural heterogeneity is set aside but they consider matching techniques to find equivalences between attributes. Table 1 illustrates the differences found between our works and others working on vertical data distribution inside polystores. ● is when the characteristic is fully presented, ◐ is when some cases are missing and ○ is when the characteristic is not addressed in the paper. We compare the considered systems inside the multi-model polystores (relational R, document-oriented D, column-oriented C and graph G), data heterogeneity (structural, semantic and syntactic), the query language(s) of the polystore and if it is question of entity class distribution in one or several system (fragmentation). In our paper, we work on relational and document-oriented system where the user is able to query a polystore in a SQL or a NoSQL language (MongoDB) in a context of vertical distribution where one entity class can be distributed in multiple databases from both systems. Our rewriting system take into account data transfer and transformation and favors the use of DBMS operators as well as its performance.

Table 1. A comparison of existing solutions on polystores

Authors	R	D	C	G	Struct.	Sem.	Syn.	Query	Entity class fragmentation
El Ahdab et al	●	●	○	○	●	◐	◐	*SQL MongoDB*	●
Barret et al [6]	●	●	○	●	○	○	●	SparkQL	○
Candel et al [5]	●	●	○	●	○	○	○	SQL	○
Ben Hamadou et al [7]	●	●	●	○	●	●	○	SQL MongoDB	○
Hai et al [8]	●	●	○	●	○	●	○	SQL JSONiq	○
Duggan et al [3]	●	●	●	○	○	●	○	Declarative	○
Forresi et al [12]	●	●	●	○	○	○	○	Spark	○

4 The proposed framework

Our framework is based on querying against unified views of a polystore. Unified views are deducted from the entity relationship model of data which highlights entity classes, attributes of entities, entity keys that can serve as distribution key, relationship roles, and relationship attributes. A logical view U_V is the factorization of all distributed entities inside the polystore, according to a fragmentation key. There is one logical view per polystore system (relational or document). We follow converting rules between the conceptual model and the logical models seen by the user: one entity corresponds to one dataset (relation or collection) and the relationships are implemented according to their cardinality. For (N,M) cardinality, in SQL a new relation is created that contains the relations keys of the N and M side (along with relationship attributes); in collections nested values are added inside the two linked collections. For (1,N) and (0,N) cardinality, in SQL a foreign key is created in the 0/1 side relation; in collections, a foreign key is created in the 0/1 side collection and nested values with foreign key are added in N-side collection. Unified views also hide data heterogeneity. To manipulate these variations of attributes, we use an existing mapping technique [7] using a dictionary for each dataset grouping for each attribute (entity, relationship) and for each key of the conceptual model, their equivalences in the unified views and in the real data implementation inside the polystore. The user can build a query against one logical view (fig. 1). We consider a user query Q_{user} on the unified view U_V referring to one DB of the polystore PL. It is composed of operators from a non-closed set of $\{\sigma\,,\,\pi\,,\,\bowtie\}$ that manipulate datasets (relations or collections). The main objective is to query the polystore by analyzing Q_{user} as an algebraic graph to generate sub queries on all systems of PL. The steps of our rewriting engine can be described by algebraic tree transformations (fig. 2): i) build an algebraic tree of the query against the unified view, ii) locate each dataset of the query in the polystore to know whose databases contains it, iii) reconstruct fragmented entities when need by adding necessary joins, iv) add transfer and/or transform operation when needed in the tree. Finally, to deal

with structural heterogeneity, the engine uses this dictionary and processes to rewrite each query operator of the algebraic tree. In case of multiple correspondences in the dictionary, our solution privileges the equivalent attribute from the same database of the system interrogated. When the final rewritten query is executed, results are presented to the user in the data format of the system selected depending on the queried logical view (relational or document).

5 Experiments

We use UniBench dataset presented in a context of multi model DBMS (http://udbms.cs.helsinki.fi/?projects/ubench). We have adapted data distribution as explained in section 2 and in fig. 2 between two SQL databases (MySQL) and one document-oriented database (MongoDB). Queries were classified according to their operators composition and to the number of dataset needed to rewrite the operation ("Monotable", "Multitable"). Our evaluation focuses on the comparison of rewriting time on each logical view (relational and document-oriented) and the impact of data distribution for one entity class per database and for one entity class in multiple databases. The join operator by itself presents the lowest rewriting time (0.0003 seconds). It is due to the presence of the entity key in every fragment inside polystores. For the selection and projection, they work more with attributes than keys. The average rewriting time of a query with a combination of all operators is higher (0.0041 seconds) than the average one for mono operator operations (0.0003 seconds). For every attribute found in the sub-queries, the dictionary is went through in order to find the exact position in the polystore and then to create the intermediate joins. Data distribution inside polystore impacts rewriting time: with two relation databases and one document-oriented one, it is easier to find the attribute in a simple structure than in nested values as we can find inside the NoSQL system. The logical view considered is the only effect to the query rewriting time in this case. Considering the execution of each query, the average execution time is close to 10 seconds, including data transformation and data transfers. Adding the rewriting time does not impact the global query time since the rewriting time does not extend 0.0041 seconds.

6 Conclusion

In this paper we focus on polystore systems with relational and document-oriented systems, where entity classes are vertically distributed between datastores and may be vertically fragmented. We define unified logical views in one data model (relational or document) that cover all the real datasets in the polystore. We define a query rewriting mechanism able to access data in all databases of the polystore according to a dictionary. Considering SPJ operators, the user can transparently query both relational and document-oriented databases with heterogeneous datasets. We have conducted experiments on a Unibench dataset, showing the effectiveness of the rewriting solution. Considering our future work

on polystore systems, we will focus on experimenting data transfers and data transformation optimisation.

Acknowledgments This work was supported by the French Gov. in the framework of the Territoire d'Innovation program, an action of the *Grand Plan d'Investissement* backed by France 2030, Toulouse Métropole and the GIS neO-Campus.

References

1. KOLEV, Boyan, VALDURIEZ, Patrick, BONDIOMBOUY, Carlyna, et al. CloudMdsQL: querying heterogeneous cloud data stores with a common language. Distributed and parallel databases, 2016, vol. 34, p. 463-503.
2. BOGYEONG, Kim, KYOSEUNG, Koo, UNDRAA, Enkhbat, SOHYUN, Kim, JUHUN, Kim, and BONGKI Moon. M2Bench: A Database Benchmark for Multi-Model Analytic Workloads. PVLDB, 16(4): 747-759, 2022.
3. DUGGAN, Jennie, ELMORE, Aaron J., STONEBRAKER, Michael, et al. The bigdawg polystore system. ACM Sigmod Record, 2015, vol. 44, no 2, p. 11-16.
4. KARNITIS, Girts et ARNICANS, Guntis. Migration of relational database to document-oriented database: Structure denormalization and data transformation. 7th international conference on computational intelligence, communication systems and networks. IEEE, 2015. p. 113-118.
5. 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.
6. BARRET, Nelly, MANOLESCU, Ioana, et UPADHYAY, Prajna. Abstra: Toward Generic Abstractions for Data of Any Model. 31st ACM International Conference on Information & Knowledge Management. 2022. p. 4803-4807.
7. BEN HAMADOU, Hamdi, GALLINUCCI, Enrico, et GOLFARELLI, Matteo. Answering GPSJ queries in a polystore: A dataspace-based approach. 38th International Conference, ER 2019, Salvador, Brazil, November 4–7, 2019, Proceedings 38. Springer International Publishing, 2019. p. 189-203.
8. HAI, Rihan, QUIX, Christoph, et ZHOU, Chen. Query rewriting for heterogeneous data lakes. 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.
9. PAPAKONSTANTINOU, Yannis. Polystore Query Rewriting: The Challenges of Variety. EDBT/ICDT Workshops. 2016.
10. GOBERT, Maxime, MEURICE, Loup, and CLEVE, Anthony. HyDRa A Framework for Modeling, Manipulating and Evolving Hybrid Polystores. IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2022. p. 652-656.
11. Chao Zhang, Jiaheng Lu, Pengfei Xu, and Yuxing Chen. 2018. UniBench: A Benchmark for Multi-model Database Management Systems. Proceedings of the Technology Conference on Performance Evaluation and Benchmarking (TPCTC 2018). Rio de Janeiro, Brazil, 7–23.
12. FORRESI, Chiara, GALLINUCCI, Enrico, GOLFARELLI, Matteo, et al. A dataspace-based framework for OLAP analyses in a high-variety multistore. The VLDB Journal, 2021, vol. 30, no 6, p. 1017-1040.