XDB in Action: Decentralized Cross-Database Query Processing for Black-Box DBMSes

Haralampos Gavriilidis[*] Leonhard Rose[*] Joel Ziegler[*] Kaustubh Beedkar[∓]
Jorge-Arnulfo Quiané-Ruiz[§] Volker Markl[*,◇]
[*]Technische Universität Berlin [∓]Indian Institute of Technology Delhi [§]IT University of Copenhagen [◇]DFKI GmbH

ABSTRACT

Data are naturally produced at different locations and hence stored on different DBMSes. To maximize the value of the collected data, today's users combine data from different sources. Research in data integration has proposed the Mediator-Wrapper (MW) architecture to enable ad-hoc querying processing over multiple sources. The MW approach is desirable for users, as they do not need to deal with heterogeneous data sources. However, from a query processing perspective, the MW approach is inefficient: First, one needs to provision the mediating execution engine with hardware resources. Second, during query processing, data gets "centralized" within the mediating engine, which causes redundant data movement. In previous work, we proposed *in-situ cross-database query processing*, a paradigm for federated query processing without a mediating engine. Our approach aims to leverage existing systems, such that it is not necessary to maintain an additional federated query engine, to increase the runtime performance and to decrease the data movement. In this demonstration, we showcase XDB, our prototype for in-situ cross-database query processing. We demonstrate several aspects of XDB, i.e. the cross-database environment, our optimization techniques, and its decentralized execution phase.

PVLDB Reference Format:
Haralampos Gavriilidis, Leonhard Rose, Joel Ziegler, Kaustubh Beedkar, Jorge-Arnulfo Quiané-Ruiz, and Volker Markl. XDB in Action: Decentralized Cross-Database Query Processing for Black-Box DBMSes. PVLDB, 16(12): XXX-XXX, 2023.
doi:XX.XX/XXX.XX

1 INTRODUCTION

Today's data is produced on different locations, and hence also stored on different DBMSes in a geo-distributed fashion. Recently, there has been an ongoing effort toward data democratization, i.e., making data more accessible [1, 7]. Inter-organizational examples include open datasets and data marketplaces while intra-organizational examples include breaking data silos across teams. This means that data scientists now have more opportunities for supporting data-driven decisions, by enriching their datasets.

However, with data spread across multiple systems, data scientists need systems that allow combining all available data efficiently. A naive approach is to load the data into a central data warehouse. However, this is not desirable, as manually crafted ETL pipelines tend to be error-prone and cannot query fresh data. In addition, it may not always be possible to centrally store all the data in a data warehouse (e.g., due to regulatory reasons [3]). Furthermore, data warehouses tend to get expensive, as one needs to provide hardware and software resources. Hence, we are looking for solutions that allow ad-hoc querying data, while automatically targeting the DBMSes that store the raw data.

The research community proposed the mediator-wrapper (MW) architecture for ad-hoc querying [6, 10] multiple DBMSes. The MW approach proposes to abstract away all the DBMSes, such that users need to only interact with one component, the mediator. From a system's perspective, the mediator is responsible for query optimization and execution. During optimization, the mediator decides which operations can be pushed down to the sources, and places the remaining operations (e.g., cross-database joins) on its mediating engine. During execution, the wrappers execute the pushed-down operations on the source and forward the results to the mediating engine to execute the cross-database operations.

From the user's perspective, the MW architecture is desirable, as it abstracts away all the complexity of dealing with different systems. Compared to the data warehousing approach, the MW architecture allows users to query underlying DBMSes ad-hoc and does not need a storage backend. However, from a query processing perspective, the MW architecture has two drawbacks. First, we need to provision the mediating engine with hardware and human resources for maintaining the software and infrastructure, leading to additional monetary cost. Second, the centralized query processing approach in the MW architecture leads to monetary cost in cloud environments, as vendors charge by network traffic, and to performance bottlenecks, as transferring data from the sources to the mediator involves costly movement operations.

Recently, we introduced *in-situ cross-database query processing*, an approach to decentrally process queries in heterogeneous geo-distributed DBMS environments [11]. We argue that cross-database queries can be executed more efficiently and with less operational cost. Our main observation is that existing DBMSes can execute all relational operations, and hence we do not need an additional meta-engine within the mediator. XDB, our proposed middleware for cross-database query processing, is only responsible for query planning. It *delegates* all operations onto the underlying systems, such that during execution all query processing operations are handled by them in a decentralized fashion, i.e., without central coordination by a mediator. After performing optimization and operator placement, i.e., deciding the mapping query plan operators to underlying systems, XDB deploys the processing operations onto the systems using views and the data movement operations using foreign (or external) tables from the SQL/MED standard.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:XX.XX/XXX.XX

The purpose of this demonstration is three-fold, i.e. to:

- illustrate the characteristics of cross-database environments by letting the audience create and interact with DBMS topologies.
- delve into XDB's internals by letting the audience interact with XDB and visualize delegation plans.
- showcase the efficiency of in-situ cross-database query processing, by comparing XDB to state-of-the-art cross-db systems.

2 CROSS-DATABASE QUERY PROCESSING

Today's data practitioners perform analytics in complex data landscapes. Data are spread across modern or legacy DBMSes, which are hosted on different on-premise or cloud environments and maintained by different entities. The research and industry communities have proposed several approaches for distributed query processing.

Federated DBMSes propose to *integrate* data across multiple heterogeneous DBMSes and data models [2, 6, 10, 18, 19]. They employ a MW architecture, where the mediator exposes a unified language and data model. However, w.r.t. query processing the MW approach has two inherent drawbacks. First, the mediator's additional execution engine introduces costs for maintaining and running the engine itself. Second, it introduces redundant data movement as all datasets are "centralized" on the mediator. This centralization results in additional costs and suboptimal runtime performance as data transfer, which dominates the execution time is also an expensive operation.

Parallel & Distributed DBMSes propose to *scale* storage and computation across multiple physical nodes, to improve performance [4, 8, 9, 16, 20]. State-of-the-art systems assume homogeneous setups, i.e. participating DBMSes are of the same type/vendor. Furthermore, such systems assume complete control over DBMSes: They decide on partitioning strategies when storing data and physical operators when processing data. While distributed DBMSes propose adequate solutions w.r.t. scalability, state-of-the-art systems do not offer out-of-the-box solutions for querying multiple heterogeneous, autonomous and geo-distributed DBMSes.

P2P DBMSes relax the assumptions about DBMS topologies. They propose query processing and data integration techniques without a central entity [5, 14, 15]. During query processing, nodes forward queries and data between them. However, P2P systems focus on lookup queries rather than complex SQL queries. Furthermore, they require installing additional communication components and also replicate data among nodes. Overall, no existing work completely addresses query processing in today's cross-db environments.

3 XDB: DECENTRAL QUERY PROCESSING

We now provide an overview of XDB [11], our system for processing cross-database queries in a decentralized manner.

3.1 Overview

We illustrate XDB's architecture and its main components in Figure 1. From a user's perspective, XDB acts as a mediator: Users submit cross-database queries, and XDB returns the result. From a system perspective, XDB is a thin middleware between users and DBMSes: It features an `Optimizer` and a `Delegator`, but does not feature an execution engine (unlike traditional MW systems). XDB performs query processing as follows: First, users send their queries to XDB through the `XDB Client` ①. Then, the `Cross-DBMS Query`

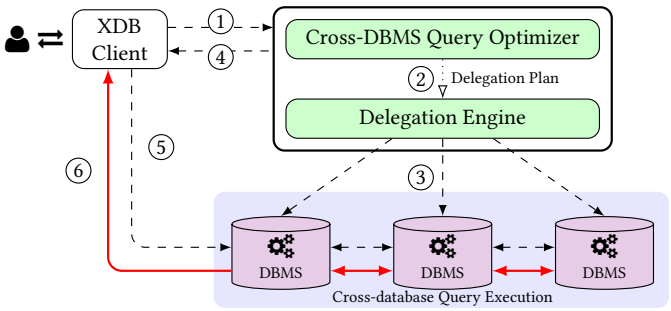

Figure 1: XDB architecture.

`Optimizer` performs logical optimizations and operator placement and outputs a Delegation Plan ②. The Delegation Plan is XDB's query plan abstraction and contains i) query execution "instructions" for each underlying DBMS, and ii) data movement operations. The `Delegation Engine` deploys the Delegation Plan onto the underlying DBMSes ③. At this point, communication only involves registering the Delegation Plan's tasks as views and movement operations as foreign tables (using the SQL/MED standard [17]) on the underlying DBMSes. During plan delegation, XDB registers a *view cascade*, i.e., views with recursive dependencies, over multiple DBMSes. After plan delegation, the Delegator returns the so-called `XDB Query` ④, a query of the form `SELECT * FROM <last view>` which will allow to recursively unfold the view cascade during execution. Finally, XDB triggers the decentralized execution by sending the XDB Query on the DBMS where it registered the `<last view>` ⑤. Evaluating the view cascade leads to a decentralized inter-DBMS query execution pipeline that computes the user results ⑥.

3.2 Cross-Database Optimization & Delegation

The goal of our optimizer is to derive an optimal Delegation Plan, which is XDB's query plan abstraction. In the following, we briefly describe XDB's Delegation Plan and our optimization process.

Delegation Plan: A Delegation Plan describes the *order*, the *placement*, and the *movement types* between operators. Essentially, a Delegation Plan is a directed acyclic graph, where nodes resemble tasks and edges data movement. A task consists of an algebraic expression that corresponds to a segment of the query plan and an annotation. The annotation defines the DBMS that will evaluate a task's expression. An edge defines the type of data movement between two tasks. We consider two movement types: implicit, where data is pipelined and explicit, where data is materialized. To create an optimal Delegation Plan, we employ a three-phase optimizer.

Three-phase Optimizer: The goal of XDB's optimizer is to generate a Delegation Plan that reduces the overall execution cost. The challenge in our environment is the extremely large search space, as we must consider all combinations for operation order, placement, and data movement. To simplify the search space, we employ a three-phase optimizer, as shown in Figure 2: In the first phase, we perform logical optimizations, i.e., query rewrites such as operator push-downs and join reordering. In the second phase, we perform operator placement. For that, we traverse the tree in a depth-first manner and annotate all operators with the base table annotation, until reaching cross-database operators. Then, for each

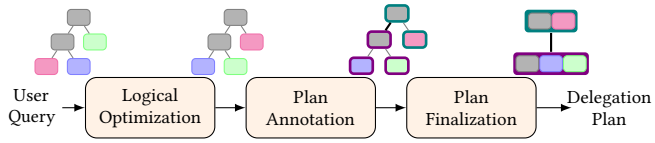

Figure 2: XDB's Optimization Pipeline.

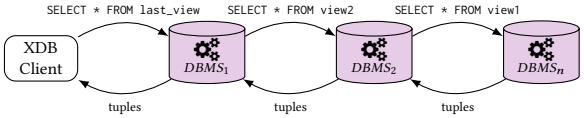

Figure 3: In-Situ Cross-Database Query Execution.

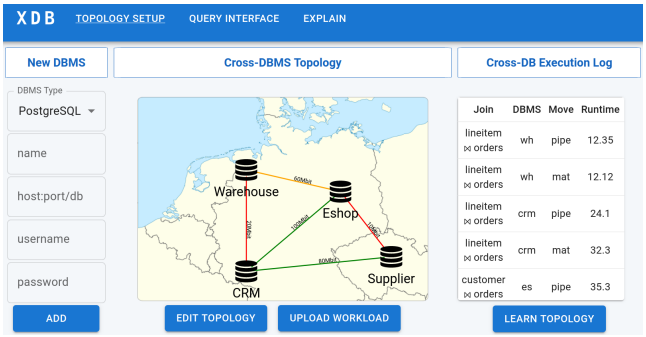

Figure 4: Cross-Database topology setup interface.

cross-database operator, we decide its placement and data movement. After annotating all operators, in the third phase, we create the Delegation Plan by grouping operators with the same annotation into tasks and connecting them through movement operations.

Placement: In our case, operator placement refers to assigning (cross-database) operators on DBMSes. What makes placement challenging is XDB's environment, i.e., a topology that consists of heterogeneous black-box DBMSes (w.r.t. vendor and available physical operator sets) running on heterogeneous physical nodes (w.r.t. hardware and network properties). In particular, costing operators in such a heterogeneous and black-box DBMS environment is not straightforward. Thus, in this demonstration, instead of costing individual operators, we treat placement as a classification task where the prediction indicates the DBMS an operation such be placed. We train our model in an offline fashion, and utilize user-provided sample cross-database queries on the underlying DBMSes that collect runtime information to generate training data. After investigating several methods, we chose random forests, as they are easy to build, and provide explainable predictions. XDB's optimizer consults the classifier during operator placement.

Delegation: After optimizing the Delegation Plan, XDB deploys it onto the underlying DBMSes. Therefore, we traverse the Delegation Plan and translate it to DBMS-specific SQL statements. We translate tasks to views, and task dependencies, i.e., data movement operations, to (SQL/MED) foreign tables. Essentially, we recursively register views on top of foreign tables, until the last task is registered. After the Delegator has finished translating and deploying the plan, it returns the XDB Query, i.e., `SELECT * FROM <last view>`.

3.3 Cross-Database Execution

After deploying the delegation plan, XDB enters the execution phase, illustrated in Figure 3. XDB triggers the execution by sending the XDB Query to the DBMS of the lastly registered view. To evaluate this view, the DBMS joins a local and a foreign table, which leads to evaluating a view on another DBMS, and so on, until the firstly registered view is reached. Essentially, evaluating the view cascade leads to a decentralized execution pipeline over multiple DBMSes. Note that during execution each DBMS's optimizer may further optimize the query locally, e.g. choose an index.

For more details on XDB, we point readers to previous work [11]. Ongoing work includes exploring optimization techniques for cross-db and composable DBMS [12] environments, and investigating optimizations for data science pipelines on multiple DBMSes [13].

4 DEMONSTRATION SCENARIOS

The goal of our demonstration is threefold. First, to show the characteristics of cross-database environments, and demonstrate the challenges w.r.t. query processing. Second, to demonstrate how XDB addresses the aforementioned challenges by exposing internal functionality w.r.t. query processing. Third, to let users interact with XDB, and showcase its advantages compared to state-of-the-art systems. For this demonstration, we developed a GUI that allows users to create cross-database topologies with DBMSes, interact with XDB by sending and explaining queries, and compare XDB's performance to state-of-the-art federated query engines. We unfold the demonstration by impersonating an organization's imaginary data team and guiding the audience through the GUI.

4.1 Plug (DBMSes) & Play (Cross-DB Queries)

As a first step, the data team registers its available DBMSes and defines the network topology. Therefore, the data team uses the GUI, as shown in Figure 4. We will provide existing docker containers running multiple systems, e.g., PostgreSQL, MariaDB, Hive, DB2, Oracle, or MSSQL with different datasets from the TPC-H benchmark. After adding the DBMSes, XDB assumes a mesh network topology, i.e., all DBMSes (nodes) are interconnected with network links (edges). However, the data team might have some DBMSes in intranets that can only be "connected" to specific DBMSes. Therefore, the data team can refine the topology, i.e., add and remove edges between DBMSes. After that, users see a catalog with the available datasets and a topology of arbitrarily interconnected DBMSes, which XDB takes into account during optimization.

4.2 *Learning* the Topology

After defining the topology, it is time for the data team to "tune" XDB's optimizer. Therefore, one can either use existing training data or generate new by providing cross-database workloads. Using this data, a Learning Component trains a random forest classifier, which is then used by XDB's optimizer during operator placement. Users can view the updated topology and explore the workload execution log (Figure 4, right). The execution log is used as a training dataset and includes features for training the classifier, i.e., data and query runtime characteristics of cross-database operations.

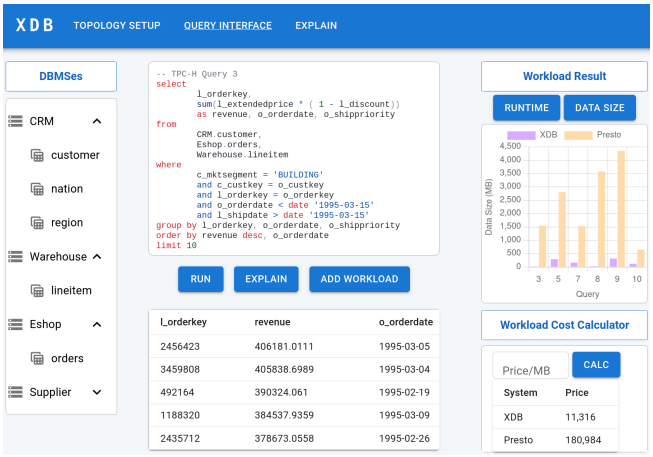

Figure 5: XDB Query Interface.

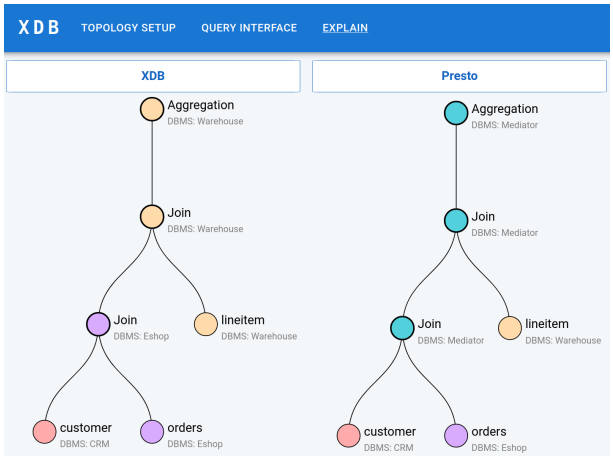

Figure 6: Cross-DB "explain": colors indicate op-placement.

4.3 Running & Explaining Cross-DB Queries

After configuring and learning the topology, it is time for the data team to interact with XDB. Users can write cross-database queries in the Query Interface pane (shown in Figure 5), or pick existing queries. To understand how queries are executed, we expose the *cross-database explain* functionality. Similar to conventional explain commands, XDB returns its query plan. As we are in a cross-database environment, XDB shows the Delegation Plan, i.e., a plan where all operators are annotated with the DBMSes the Optimizer assigned them to (Figure 6, left). To show how XDB's optimization process differs from typical MW systems, we also visualize MW query plans (Figure 6, right). To see how different topologies influence optimization, users can interactively change the topology and rerun the cross-database queries.

4.4 KPIs: Runtime & Data Transfer Performance

After interacting with XDB through ad-hoc queries, the data team explores the overall applicability and operational costs of XDB. In particular, the team is interested in the potential performance gain by switching to XDB, with their KPIs being runtime performance and data transferred during execution. Therefore, users can upload their workloads to our benchmarking component, or utilize existing workloads. The benchmarking component then runs the workloads on XDB and state-of-the-art MW systems (e.g., Presto), and generates a performance report. The report features two main metrics: runtime and data transfer. Furthermore, the GUI features a price/MiB calculator, for estimating monetary cloud vendor costs for each approach.

ACKNOWLEDGEMENTS

This work was funded by the German Ministry for Education & Research as BIFOLD - Berlin Institute for the Foundations of Learning & Data (BIFOLD22B; BIFOLD23B), and by Software Campus as PolyDB (01IS17052). We thank Anastasios Tsigkros for bootstraping XDB's workload learning component. This work is dedicated to the memory of Jorge-Arnulfo Quiané-Ruiz.

REFERENCES

[1] Daniel Abadi, Anastasia Ailamaki, David Andersen, Peter Bailis, Magdalena Balazinska, et al. 2022. The Seattle report on database research. In *CACM*.
[2] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, et al. 2015. Spark sql: Relational data processing in spark. In *SIGMOD*.
[3] Kaustubh Beedkar, Jorge-Arnulfo Quiané-Ruiz, and Volker Markl. 2021. Compliant Geo-distributed Query Processing. In *SIGMOD*.
[4] Philip A Bernstein, Nathan Goodman, Eugene Wong, Christopher L Reeve, and James B Rothnie Jr. 1981. Query processing in a system for distributed databases (SDD-1). In *TODS*.
[5] Peter Boncz and Caspar Treijtel. 2003. AmbientDB: relational query processing in a P2P network. In *International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*.
[6] Michael J Carey, Laura M Haas, Peter M Schwarz, Manish Arya, William F Cody, et al. 1995. Towards heterogeneous multimedia information systems: The Garlic approach. In *RIDE-DOM'95*.
[7] European Commission, Content Directorate-General for Communications Networks, Technology, E Scaria, et al. 2018. *Study on data sharing between companies in Europe : final report*. Publications Office. https://doi.org/doi/10.2759/354943
[8] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, et al. 2013. Spanner: Google's globally distributed database. In *ACM TOCS*.
[9] Umur Cubukcu, Ozgun Erdogan, Sumedh Pathak, Sudhakar Sannakkayala, and Marco Slot. 2021. Citus: Distributed PostgreSQL for Data-Intensive Applications. In *SIGMOD*.
[10] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, et al. 1997. The TSIMMIS approach to mediation: Data models and languages. In *Journal of intelligent information systems*.
[11] Haralampos Gavriilidis, Kaustubh Beedkar, Jorge-Arnulfo Quiané-Ruiz, and Volker Markl. 2023. In-Situ Cross-Database Query Processing. In *ICDE*.
[12] Haralampos Gavriilidis, Lennart Behme, Sokratis Papadopoulos, Stefano Bortoli, Jorge-Arnulfo Quiané-Ruiz, and Volker Markl. 2022. Towards a Modular Data Management System Framework. In *CDMS @ VLDB*.
[13] Yordan Grigorov, Haralampos Gavriilidis, Sergey Redyuk, Kaustubh Beedkar, and Volker Markl. 2023. P2D: A Transpiler Framework for Optimizing Data Science Pipelines. In *DEEM @ SIGMOD*.
[14] Alon Y Halevy, Zachary G Ives, Jayant Madhavan, Peter Mork, Dan Suciu, and Igor Tatarinov. 2004. The piazza peer data management system. In *IEEE TKDE*.
[15] Ryan Huebsch, Joseph M Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. 2003. Querying the Internet with PIER. In *VLDB*.
[16] Bruce G. Lindsay. 1987. A retrospective of R*: a distributed database management system. *Proc. IEEE* 75, 5.
[17] Jim Melton, Jan Eike Michels, Vanja Josifovski, Krishna Kulkarni, and Peter Schwarz. 2002. SQL/MED: a status report. In *ACM SIGMOD Record*.
[18] Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, et al. 2019. Presto: Sql on everything. In *ICDE*.
[19] Jeff Shute, Radek Vingralek, Bart Samwel, Ben Handy, Chad Whipkey, et al. 2013. F1: A distributed SQL database that scales. In *VLDB*.
[20] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, et al. 2020. Cockroachdb: The resilient geo-distributed sql database. In *SIGMOD*.