Execution of Analytical Queries Over Big Data

Bruno Silva, *Mestrado em Engenharia informática, Seminário*

Abstract— The growing variety of databases with different data models, query languages, architectures and other characteristics generates a great deal of variety. Different managements teams of each database system generate different representations in schemas to store the same type of information, generating even more heterogeneity. These issues introduce difficulties in integrating data from different database systems in order to perform data analytics. The need for analytical queries for near real time data is rising in the area of IoT and traditional data warehouses do not provide recent data, therefore an alternative would be to query data where it lives. Several architectures and concepts have been discussed in the literature throughout years, with recent platforms being capable to query big data from distributed and heterogeneous databases, such as Presto. This platform provides a good starting point for future work to be developed for the thesis: present inter relations between tables as if data is on a single database.

Index Terms— Database integration; Distributed databases; Heterogeneous Databases;

1 Introduction

Throughout time, multiple advancements have been made in the database systems field. Initially, relational databases were prominently used by applications to store data of various types, such as user accounts and bank accounts. This type of database is great to store information in a centralized, structured and consistent way. [1].

In late 2000's and early 2010's, the amount of data that needed to be processed or stored was "too big, too fast or too hard for existing tools to process", designated by Big Data [2]. In this context, the disadvantages of relational DB's emerged, namely the fact that these databases were centralized and could not scale. Relational DB's also force data to have a specific structure. Nowadays, companies have interest in storing data that is semi-structured or unstructured, such as image or videos [1], but also data produced by sensors or machines. According to [1], "a very large portion of the data (approximate 70% to 90%) in the business world today is unstructured data, while SQL can only handle the structured portion of data".

Several types of databases were later created that addressed relational DB's disadvantages, such as higher availability, storage of unstructured data and horizontal scalability. These databases are called NoSQL (Not Only SQL) and the philosophy behind them is that a state of consistency is not necessarily guaranteed after each transaction (data may be temporarily inconsistent or stale), and it is instead preferred that these databases have a better performance on data handling and eventually become consistent in order to improve availability [3].

There are several types of NoSQL databases, each one used for a specific purpose, and a database must be chosen depending on the type of data and context in which information will need to be used. In a big platform or company, several databases are used for specific domains of the platform or parts of the business because those databases are more suitable. An ecommerce website, for example can use a key-value database to store information regarding a shopping cart and a document database to store the catalog of different products with distinct attributes [1].

Companies use the most appropriate database to store data from different parts of the business. Therefore, data is scattered between different databases and sometimes, there is a need to analyze data stored in different database systems.

The use of different types of databases inside the same application system is called polyglot persistence [1]. To say that these databases are heterogenous means that each database has its own DBMS with specific query languages, schema, and database type (relational, key-value, document). It must also be considered that companies can have several databases geographically scattered, storing data for specific regions. Dealing with distributed databases means dealing with possible replicated or fragmented data and communication over the web.

Over the years, several solutions and frameworks have been developed to integrate multiple distributed and possibly heterogeneous databases. The proposed work for this thesis is to search and explore existing tools and frameworks and understand their capability to query heterogenous and distributed databases while also supporting big data. After a framework is selected and its capabilities determined, functionalities should be built on top of the framework so that the requirements for the thesis are met. The final platform should allow users to submit queries and allow joins and aggregate operations from data originated from heterogenous databases, thus making the physical distribution and its heterogeneity transparent to the user.

2 Heterogeneous and distributed databases: an overview

The use of different database systems (DBS) for a specific domain and target application is now a standard even for medium or small companies or platforms because there are many databases with different perks and advantages that suit different objectives [1]. But the use of multiple heterogeneous databases introduces new challenges, such as the use of different schemas between the databases and distinct ways to query each database due to semantics of query languages [4]. Distributed databases add the problem of communication over the web with each database.
Ideally, one would like to query heterogenous databases and gather information from those DB's by using aggregators or joins in order to integrate the data and therefore retrieve more value from all the information regardless of the schema and database specifications such as language or underlying architecture.

2.1 Challenges to overcome with database integration

One of the challenges when retrieving data from autonomous database systems is the query language and data models of each DBMS. But there are other issues to consider, usually classified in 3 categories: distribution, heterogeneity and autonomy [4][5].

2.1.1 Heterogeneity

Heterogeneity can be due to various factors in DBSs: data model (relation, graph, key-value), query language, software and capabilities of the DBMS, etc…. The proliferation of multiple different database systems contributed for these types of heterogeneities. But even if the same DBS is used, heterogeneity can still occur in the schema, which causes syntactic and/or semantic heterogeneity at the level of the data model.

For example, considering a company that has multiple subsidiaries with the same type of relational database system, each one containing information regarding its employees, sales and other business-related information. Since each subsidiary is going to implement their own local database, the schema will surely have differences between each subsidiary, unless the company forces all of them to adopt a specific data model. The most likely scenario is to end up with different schemas to store the same type of information per subsidiary. If someone in the company's headquarters tries to integrate this information from each database by querying them all with multiple different database schemas, then this is a hard task. Examples of schema mismatch across databases can be:

- Using different name variations to specify the same concept (such as a table or a table attribute)
- Using disparate data types for the same data domain (such as using varchar or numeric for the attribute 'employeeID'), table attributes, or overall table organization (such as missing table attributes and general mismatch in table structure)
- Using uneven scales for certain measurements in attributes. [6]
- Attributes with same syntactic meaning can have different semantic meaning. For example, one attribute in DB1 which stores averages of production cost per day without taxes and another attribute in DB2 which also stores production cost per day by with taxes [6]

To understand that two components in different database schemas are semantically related despite these differences is known as schema matching [4].

2.1.2 Distribution

Distribution refers to the physical distribution of data that may be in DBSs. These DBSs can be geographically distributed or distributed between machines [5].

2.1.3 Autonomy

Autonomy defines the amount of operations related to data access, configuration or development [4] that is possible to perform in a DBMS, or in other words, "the degree to which individual DBMSs can operate independently" [4]. A DBS with no autonomy means that that it cannot receive requests directly from the users or client applications, i.e., requests must be sent through a mediator. The level of autonomy of a DBS can also be variable, with higher autonomy DBSs allowing the execution of more operations than lower autonomy ones. A database system with high autonomy generates more heterogeneity [5].

2.2 Database integration

There are 2 ways for the integration of data from multiple distributed and possible heterogenous databases: the materialized approach or the logical approach [4][7].

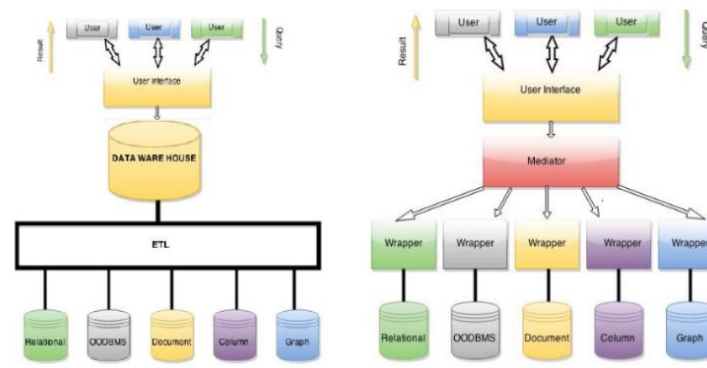

Fig. 1. Approaches for database integration. Materialized approach on the left and logical approach on the right. Adapted from [7]

The data Materialization approach, depicted in Fig. 1 on the left, is currently the most used by enterprises for the implementation of decision support systems. To extract data from each database belonging to the company, scripts are created for each one and that data is then extracted, transformed and loaded (ETL) into one single relational database, called data warehouse [7]. This database can support large volumes of data and is organized in such a way that allows users to perform analytical queries of different data and in different time periods.

However, ETLs can be costly and take a long time to integrate all data. Data is extracted from each source periodically which means the data in the data warehouse may not be up to date. This approach also generates data redundancy because we have the same information in each database and in the data warehouse, even if organized differently in the latter.
Nowadays, in the IoT area, there are several sensors that are constantly generating data. IoT projects need to make analytical analysis on the most recent data, or near real time analysis. The logical approach, depicted in Fig. 1 on the right, provides an alternative that might suit these needs, as we directly query and integrate only the data needed for the analytical queries directly from each source database [7]. To do this, a reference architecture is commonly referred in the literature ([4, 5, 7]) as the mediator/wrapper. The base idea is that a mediator is able to receive a query that references data in multiple databases, and it is able to decompose the query and solve schema heterogeneities through a unified schema of all databases. The queries are forwarded to wrappers that are responsible to communicate with a database and directly query and extract data from it [7]. A DBMS will manage the access to each database by employing techniques to handle the distribution and heterogeneity of each database, making the heterogeneities, autonomy and distribution of each databases transparent to the user.

A problem with this approach is that having local users that are using the database for its intended purpose (i.e. a costumer consulting the list of flight schedules) and global users issuing analytical queries to these database at the same time can degrade the performance of each database and give to the end user an overall unpleasant experience on the platform because the database operated slower [4].

The thesis focuses on developing a system that logically integrates data, therefore the state of art for the thesis and this article is about concepts and systems that can logically integrate data, using variations of the mediator/wrapper approach.

2.3 Multidatabase Systems

These systems use the mediator/wrapper architecture discussed earlier, with the mediator employing the global schema that solves the schema heterogeneities.
It is worth noting that, while searching through the state of art, a few ambiguities between terms were found. Different authors use distinct taxonomies and terms to identify the different types of multidatabase systems ([4], [6], [5]), therefore, it was decided to adopt the taxonomy detailed in [6] and also used in [5].

2.3.1 Federated multidatabase systems

A federated database management system (FDBMS) is constituted by multiple autonomous DBSs, each one with its own schema. A single database system can simultaneously participate in one or more federations and continue to carry its local operations [6]. The federated database system (FDBS) consists of the FDBMS and the local DBSs. The global schema contains schema mappings between tables in the form of views [6]. In order to make the conceptual schema of an individual database accessible to the FBMS, each DBS must supply a schema with the information that can be accessed by the federation in that DB, allowing database managers to select which data can be made available for the federation [4, 7]. The schemas of each database are then integrated in a federation schema located in the FDBMS and contains a unified global schema with all heterogonies solved and information regarding the mapping of data and tables to databases and their location [6]. The federated schemas are then used to generate an external schema to be used by each user/application of a federation. These schemas can be different for each users/applications because they may want different information from the federation and external schemas allow control of information in the federated schema(s) [6].
There are two variations of FDBMS:

- Loosely coupled FDBMS: in this scenario, each administrator of each database must create the federated schema and make it available to the federation [7, 5].
- Tightly coupled FDBMS: the administrator of the FDBMS is responsible to create the federated schema for the federation [5, 7].

2.3.2 Non-federated multidatabase system

Non-federated databases differ from the federated databases by the level of autonomy in each database system, which is lower or (even non existing) than the autonomy of the database systems in a federation database [6][5]. Some authors refer to these databases as simply multidatabase systems ([4, 5]).

2.4 Polystore Systems

Multidatabase systems were the first to be discussed and conceptualized in the literature (back in 1990) and did not considered multi data model querying because all databases were of the same data model: relational. Current systems (2010 and beyond) employ additional implementations to deal with multi data model querying, while also solving schema heterogeneities. To distinguish a system that can only integrate data from relational databases and a system that can integrate data from databases with heterogeneous models we say that the former are multidatabase systems and the latter are polystore systems [8].
Polystore systems may also have more than one query interface [9], such as SPARK SQL which allows the use of SPARK-like statements mixed with SQL statements inside [9].

3 Frameworks and Tools for Heterogeneous Databases and Big Data

3.1 MDB1 and MDB2 multidatabase systems.

In 2001, the De La Salle University developed 2 multidatabase platforms: MDB1 and MDB2. The description of the platforms is entirely based on [10].
Both these platforms have a user interface for users to manually and visually perform schema integration across

databases with different schemas, which means users can solve the schema matching problems specified in section 2.1 in a more efficient and less error prone way.
The first one, named MDB1, is a federated database system that contains a global schema of all databases and periodically pulls the data from each database and stores all data in a global database using a global schema, integrating all data and solving heterogeneities using rules specified by the user. One major drawback from this approach is that the data is not up-to-date in the global database and that database might not be able to hold all the data.

The second system, MDB2, fetches data directly from each database when needed and only what is needed by dividing a query into a sub-query to each DBS. For example, if a user wants to aggregate data from DB1 and DB2 the system decomposes the query into sub queries and translates them using the global schema for each DB to fetch necessary data from DB1 and DB2 and then performs aggregation on an integrator DB, returning the results to the user. MDB2 has no query optimizer.

3.2 BigDAWG polystore system

BigDAWG (Big Data Working Group) is an open source polystore system developed by Intel Science and Technology Center for Big Data (ISTC) and follows the mediator/wrapper architecture. It uses a concept of islands as mediators [8]. Islands contain data model information, schema integration, query language in which users issue queries to management systems in the island for query execution on one or more DBSs transparently [8]. Ideally, users should configure islands for each type of data model they want to query and connect database to islands according to their data models, but it is possible to connect DBSs that are of different data models if necessary to integrate data from that database as well. To connect a database to an island, a shim is written and used as the wrapper to translate a query to the language of a target database system [8].

If a user wants to query databases that are on the same island (single island query), then users must specify that in the query language using the 'cast' operator of the BigDAWG query language to convert data between databases. This forces users to know details regarding the underlying databases and the organization in each island.

3.3 Apache Drill

A more recent architecture of a logical data integration platform is Apache Drill, which is a distributed query engine and ready to scale. Distributed because it is designed to be installed on a cluster of machines but can also run on only one. It provides an SQL query interface that most developers are familiar with. The architecture of Drill is represented in Fig. 3 and is comprised by Drillbits, an installation of Drill on a machine that handles the query (acts as mediator), and plugins that connect to DBSs (act as wrapper). When a user issues a query, Zookeeper is responsible for assigning an available Drillbit to receive the query, becoming the foreman. The latter is responsible to process, decompose and generate an optimized plan for the other Drillbits to execute each subquery resulting from the query decomposition [9]. A subdivision of a query that agglomerates data from 2 different databases can result in 2 subqueries: one to fetch data from a database and another to fetch data from the other database. A Drillbit then uses a plugin to communicate with the respective database and fetch the necessary data [11]. The Foreman Drillbit finishes any processing and returns data to the user.

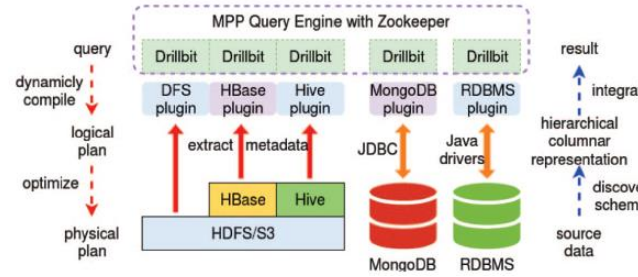

Fig. 3. Apache Drill Architecture. Taken from [7]

3.4 Presto

Presto is also a distributed query engine capable of processing big data developed by Facebook and is open source. Users use SQL as the query language to issue queries to the system. It functions as a master-slave, the master being called coordinator, and slaves called workers. The coordinator is responsible for receiving the query and then parsing it, planning and optimizing it as well [12]. The coordinator creates subqueries and generates a distributed plan to assign subqueries to workers which are then responsible to fetch data from data sources [12]. Workers can cooperate in order to support the processing of more than one query simultaneously [12]. Connectors are the wrappers that allow communication with a specific data source. They are extensible because they can be created and modified by anyone. Figure 4 illustrates the architecture of Presto.

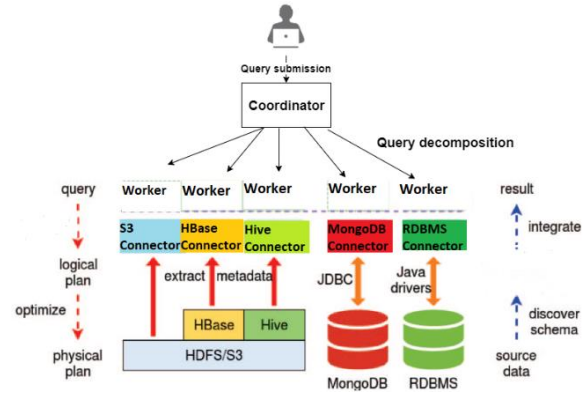

Fig. 4. Presto Architecture. Adapted from [7]

3.4.1 Presto Experiments

Presto has capabilities to query big data from heterogenous and distributed databases and supports a wide variability of databases through connectors (more than Apache Drill's plugins). Performance-wise, Presto was considered faster than Apache Drill and 2 other similar applications by an experiment conducted in [2], and [13]

considered Presto as the most effective tool in terms of generalization (SQL support, performance and functionalities), therefore surpassing Apache Drill by comparison. These reasons led to further experiment and testing of Presto.

Presto version 319 was installed and configured with only one coordinator (also operating as a worker) and dummy tables on PostgresSQL were created to query data in order to learn the basics of Presto. After that a more complex experiment was made. On a local machine, Presto Coordinator and MongoDB 4.0 were installed, and on a remote machine a Presto Worker and PostgresSQL 11 database were installed. Sample databases were created using TPC-DS.

After the necessary configurations, Presto was able to use the coordinator and the worker installed on the machines to access data in the DBs. The queries made from Presto to PostgresSQL were tracked by consulting the logs available in the database, hoping that this could provide more insight as to how presto operates.

After these experiments, a few things were noticed:

- Presto may not correctly update its schema of all fields of a mongo collection.
- Presto executes simple queries to each database (select and filters) and joins and aggregations are made on presto itself, as mentioned in [12]. This can be inefficient because we could filter even more the data.

4 Future work

After the previous experiments and conclusions, it was decided that Presto would be an ideal tool to use for the thesis. In addition to the problems specified previously, Presto is not capable of establishing inter relations between tables, which is a necessity in order to provide the level of abstraction required for this thesis. To do this, further state of the art and techniques must be studied.

The architecture for the platform to be developed should use Presto to gather information from the databases. Additional layers will be built on top of Presto, as specified in Fig. 5. These layers should provide an interface in which the users send queries in MDX (further details of the language to be defined) and another layer receives those queries and uses Presto to retrieve data and establish inter relations between the data. Additionally, optimization mechanisms could be implemented, such as prefetching of data and cache of metadata to speed up data extraction and inter relation assignment.

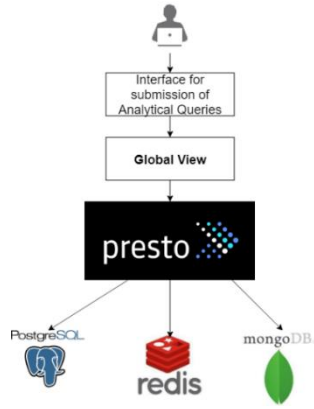

Fig. 5. Future work architecture example.

5 Conclusion

After having researched and learned about the state of art for distributed and possibly heterogenous database integration, several tools and references were found to address the specified issue in a variety of different ways. In the end, a tool to use as basis for the thesis was found, providing a good starting point for future work, as Presto fulfills some of the essential requirements for this thesis such as querying and aggregating data from heterogeneous DBSs and over big data.

The overall idea of this work is not to completely replace data warehouses for analytical querying and decision making, but to instead provide an alternative method to data analytics that better suits some use cases, such as the need for near real time data analytics in the area of IIoT.

References

[1] P. P. Khine and Z. Wang, "A review of polyglot persistence in the big data world," *Inf.*, vol. 10, no. 4, 2019.

[2] M. Y. Santos *et al.*, "Evaluating SQL-on-Hadoop for Big Data Warehousing on Not-So-Good Hardware," vol. 17 [Online]. Available: http://dx.doi.org/10.1145/3105831.3105842. [Accessed: 08-Nov-2019]

[3] J. D. Cook, "database transactions," 2009. [Online]. Available: https://www.johndcook.com/blog/2009/07/06/brewer-cap-theorem-base/. [Accessed: 08-Nov-2019]

[4] M. Tamer Özsu and P. Valduriez, "Principles of Distributed Database Systems, Third Edition," 2011.

[5] M. Chen and R. Hofestädt, "Approaches in Integrative Bioinformatics Towards the Virtual Cell."

[6] A. P. Sheth and J. A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," *ACM Comput. Surv.*, vol. 22, no. 3, pp. 183–236, Jan. 1990.

[7] B. Garg and K. Kaur, "Integration of heterogeneous databases," in *Conference Proceeding - 2015 International Conference on Advances in Computer Engineering and Applications, ICACEA 2015*, 2015, pp. 1033–1038.

[8] J. Duggan *et al.*, "The BigDAWG Polystore System."

[9] R. Tan, R. Chirkova, V. Gadepally, and T. G. Mattson, "Enabling query processing across heterogeneous data models: A survey," *Proc. - 2017 IEEE Int. Conf. Big Data, Big Data 2017*, vol. 2018-Janua, pp. 3211–3220, 2017.

[10] C. Y. Ko, "THREE APPROACHES TO A MULTIDATABASE SYSTEM," 2001.

[11] "Apache Drill - Architecture Introduction." [Online]. Available: https://drill.apache.org/docs/architecture-introduction/. [Accessed: 06-Dec-2019]

[12] R. Sethi *et al.*, "Presto: SQL on everything," *Proc. - Int. Conf. Data Eng.*, vol. 2019-April, pp. 1802–1813, 2019.

[13] A. Kolychev and K. Zaytsev, "RESEARCH OF THE EFFECTIVENESS OF SQL ENGINES WORKING IN HDFS," *J. Theor. Appl. Inf. Technol.*, vol. 31, p. 20, 2017 [Online]. Available: www.jatit.org. [Accessed: 08-Dec-2019]