Modern Federated Database Systems: An Overview

Leonardo Guerreiro Azevedo, Elton Figueiredo de Souza Soares, Renan Souza
and Marcio Ferreira Moreno
IBM Research, Brazil

Keywords: Federated Database, Polyglot Database, Multistore, Polystore, Multidatabase, Heterogeneous Data Stores, NoSQL, Dbaas, Distributed File System, Data Processing Frameworks.

Abstract: Usually, modern applications manipulate datasets with diverse models, usages, and storages. "One size fits all" approaches are not sufficient for heterogeneous data, storages, and schemes. The rise of new kinds of data stores and processing, like NoSQL data stores, distributed file systems, and new data processing frameworks, brought new possibilities to meet this scenario's requirements. However, semantic, schema and storage heterogeneity, autonomy, and distributed processing are still among the main concerns when building data-driven applications. This work surveys the literature aiming at giving an overview of the state of the art of modern federated database systems. It presents the background, characterizes existing tools, depicts guidelines one should follow when creating solutions, and points out research challenges to consider in future work. This work gives fundamentals for researchers and practitioners in the area.

1 INTRODUCTION

Several modern applications manipulate diverse datasets with different models and usages, *e.g.*, medical informatics, intelligent transportation, etc. "One size fits all" is not effective in such scenarios. The use of a single database and a unique data model for all data in different data models may degrade performance and executing ETL (Extract-Transform-Load) processes to load all data in a single database may be very expensive (Stonebraker et al., 2007). Besides, manual data curation and maintenance of the ETL pipelines (due to adaptations caused by, *e.g.*, domain evolution) are labor-intensive (Tan et al., 2017) (Bondiombouy and Valduriez, 2016) (Stonebraker, 2015).

The problem of accessing heterogeneous data sources has been studied in the context of multidatabase and data integration systems (Kolev et al., 2016a). Several new data management solutions have emerged, such as distributed file systems (*e.g.*, GFS[1] and HDFS[2]), NoSQL data stores (*e.g.*, MongoDB, Allegrograph, Neo4J, Titan, Dynamo, BigTable, Redis) and new data processing frameworks (*e.g.*, Spark) as well as hybrid (multimodal, *e.g.*, OrientDB, ArangoDB, or NewSQL, *e.g.*, Google F1, LeanXcale). The RDBMS (Relational Database Management System) has been evolved to manage different kinds of data (*e.g.*, multimedia objects, XML documents, spatial data), like IBM DB2[3] which was built on a standard SQL engine, but it has evolved to be a hybrid data management system for structured and unstructured data. Usually, using one single DBMS results in loss of performance and flexibility for specific applications. For instance, a column-oriented DBMS is one order of magnitude better for Online Analytical Processing (OLAP) workloads than an RDBMS (Özsu and Valduriez, 2020), while SDBMS (Stream Database Management System) is more efficient for stream data, which RDBMS does not even support (Nayak et al., 2013). Thus, a variety of data-processing architectures may be required for specialized markets (Stonebraker et al., 2007).

Schema, semantic, and data sources heterogeneity, autonomy, and distributed processing are still concerns (Tan et al., 2017). A federated system arises as a solution. It is a middleware that provides a seamless interface to heterogeneous data systems with an independent data model and (perhaps) data schemes (Stonebraker, 2015).

This work overviews the state-of-the-art of the new generation federation systems.

It is divided as follows. Section 2 presents the main concepts. Section 3 characterizes existing tools.

[1]Google File System.

[2]Hadoop Distributed File System.

[3]https://www.ibm.com/analytics/db2

Azevedo, L., Soares, E., Souza, R. and Moreno, M.
Modern Federated Database Systems: An Overview.
DOI: 10.5220/0009795402760283
In *Proceedings of the 22nd International Conference on Enterprise Information Systems (ICEIS 2020) - Volume 1*, pages 276-283
ISBN: 978-989-758-423-7

Section 4 presents guidelines and research challenges. Finally, Section 5 concludes.

2 MAIN CONCEPTS

The shift in the federation database has arisen due to the storage requirements of modern applications, which resulted in the development of several distinct storage technologies to meet specific needs. Now, it is a requirement for these technologies to work together. This section overviews the state-of-the-art.

2.1 Storage Solutions

There are three main layers of storage: distributed storage; database management; and, distributed processing (Bondiombouy and Valduriez, 2016).

Distributed Storages. Include files and objects storages. File storage works on unstructured data (*i.e.*, sequences of bytes), organizing them as fixed-length or variable-length records. The system organizes files hierarchically, and stores file metadata (*e.g.*, file name, owner, access permission) separate from content. For shared-nothing, examples are GFS, HDFS, and GlusterFS; for shared disk, an example is Global File System 2 (GFS2). Object storage stores data as an object which has a unique identifier (*oid*), properties and metadata. Examples are Lustre and XtreemFS. Also, Ceph and Ozone are systems that combine block and object storage.

NoSQL (Not Only SQL) Systems. Emphasize scalability, fault-tolerance, and availability, sometimes at the expense of consistency. The main categories are key-value, wide column, document, and graph, as well as hybrid (multimodel or NewSQL) (Özsu and Valduriez, 2020). SolidIT[4] presents a comparison of systems.

Key-value Systems: store data as key-value pairs where the key identifies the record and the value is a schemaless data. Their typical operations are `put(key, value)`, `get(key)` and `delete(key)`. Examples are Redis, Dynamo, Memcached, and Riak. Extended Key-Value systems store records (a set of key-value pairs) in collections (or domains), *e.g.*, the domain `Customers` where each customer has Customer Id, first name, last name, etc. Examples are Amazon SimpleDB and Oracle NoSQL Database.

Wide Column Systems: store data as a table but allowing nested values in a schemaless way where a column may have column values. Each column has a name, a value, and a timestamp (used for versioning). Examples are Google Bigtable, Apache HBase, Cassandra, and Accumulo.

Document Systems: are advanced key-value systems where values are of the document type, such as JSON, YAML, or XML. It stores records in collections (similar to tables). Records in a collection may have different schemes. Besides simple key-value operations, document stores offer an API or query language. Examples are MongoDB, CouchDB, Couchbase, RavenDB, and Elasticsearch.

Graph Database Systems: manipulate data as graphs. Their use has grown to manage data with inherent graph-like nature, *e.g.*, Web, geographical systems, transportation, telephones, social and biological networks. The graph database model represents schema and instances as a (labeled)(directed) graph or generalization of the graph structure (*e.g.*, hypergraphs or hypernodes) and graph integrity constraints (Angles and Gutierrez, 2008). Data are represented as nodes and edges (which connect two nodes). *E.g.*, *horse* and *apple* nodes and a `likes` edge to represent *horse* `likes` *apple*. Nodes and edges may have properties, *e.g.*, name and birthday properties for horse and color for apple. Often, these systems provide query languages that allow for graph traversals and other typical graph operations, like breadth and depth search. Examples are Neo4J, Infinite Graph, Titan, GraphBase, Trinity, and Sparksee.

Triplestores: or RDF stores, are the matter of choice for storing and querying semantic datasets (Haslhofer et al., 2011)(Iancu and Georgescu, 2018), which are often described using RDF (Resource Description Framework), a standard model for data interchange. In RDF, datasets are represented as triples (subject, predicate, object). That is a value (*object*) of a property (*predicate*) of a resource (*subject*) (Zulkefli et al., 2013). *E.g.*, `(LeonardoDaVinci, hasCreated, TheMonalisa)`. Each part is represented as a Uniform Resource Identifier (URI). RDFS (RDF Schema) and OWL (Web Ontology Language) are RDF serializable vocabularies, commonly used to represent ontologies, which define classes and attributes of URIs and their relationships (Iancu and Georgescu, 2018). Moreover, triplestores are capable of processing a large amount of RDF data (Modoni et al., 2014), handling semantic queries and using inference for uncovering new information out of the existing relations. Examples are AllegroGraph, GraphDB, MarkLogic, Mulgara, Profium Sense, Blazegraph, Virtuoso, Marmotta, Stardog, Apache Jena, RDF4 (former Sesame), Oracle Database 12c (Iancu and Georgescu, 2018).

Hybrid Data Stores: combines capabilities typically found in different data stores and DBMS. They

[4]https://db-engines.com/en/ranking

may be multimodel NoSQL systems, which combines multiple data models (examples are OrientDB, ArangoDB, and Microsoft Azure Cosmos DB), and NewSQL DBMSs, which combines the scalability of NoSQL with the strong consistency and usability of relational DBMS (examples are Google F1, LeanXcale, Apache Ignite, among others). Hybrid Transaction and Analytics Processing (HTAP) is a class of New SQL aiming at performing OLAP and OLTP in the same data allowing real-time analysis and avoiding ETL processing.

Data Processing Frameworks. Handle a high volume of data in real-time (Zheng et al., 2015). They focus on data analysis to increase understanding, pattern discovery, and gain insights. They handle data in batches, in a continuous stream or both ways (Gurusamy et al., 2017). Typically, those systems support operators that are automatically parallelized (Bondiombouy and Valduriez, 2016). Examples are (Gurusamy et al., 2017): (i) Batch-only: Hadoop MapReduce; (ii) Stream-only: Apache Storm and Apache Samza; (iii) Hybrid: Apache Spark and Apache Flink.

2.2 Taxonomies

There are two main taxonomies to classify federated data systems.

Tan *et al.* proposed a taxonomy of four categories considering heterogeneity in data stores and query interfaces (Tan et al., 2017):

- **Federated Database System.** *Homogeneous data stores* and *single standard query interface*. They feature mediator-wrapper architecture and employ schema-mapping and entity-merging techniques for data integration. Semantics heterogeneity is a challenge. Example: Multibase.
- **Polyglot System.** *Homogeneous data stores* and *multiple query interfaces*. Different query interfaces provide semantics, which significantly simplifies query formulation. Example: Spark SQL allows access to data in relational and procedural modes.
- **Multistore System.** *Heterogeneous data stores* and *single query interface*, categorized as:
  - Systems that integrate distributed file systems with RDBMs, such as HadoopDB, Polybase, and JEN.
  - Systems that integrate NoSQL systems with RDBMSs, such as BigIntegrator, Forward, and D4M.
  - Systems focused on optimizing data placement across data stores for query performance, such as ESTOCADA, Odyssey, and MISO.
  - Systems that adopt ontologies and apply semantic approaches (schema-mapping and entity-resolution techniques) to mediate relational and non-relational data sources, such as TATOOINE and OPTIQUE.
- **Polystore System.** *Heterogeneous data stores* and *multiple query interfaces*, categorized as:
  - Systems focused on query answering, such as: BigDAWG, Myria and Apache Drill.
  - Systems that concentrate on multi-platform data-flow scheduling and analytics, such as QoX, Musketeer, and Rheem.
  - Systems focused on data ingestion and derivation with heterogeneous data stores, such as AWESOME.

Bondiombouy and Valduriez's classification is based on the data coupling of the systems (Bondiombouy and Valduriez, 2016). They base their work in cloud data stores and call them as multistore. Multistore is a system that provides integrated access to several data stores, such as NoSQL, RDBMS, or HDFS, sometimes through a data processing framework. Multistores can be classified as:

- **Loosely Coupled System.** *Autonomous local data stores accessed by a common language or by their local language*. Examples: BigIntegrator, Forward, and Qox.
- **Tightly Coupled System.** *Local data stores accessed by the multistore system using a single language for querying structured and unstructured data*. They aim at efficient querying for (big) data analytics and/or self-tuning. Examples: Polybase, HadoopDB, Estocada, Odyssey, and JEN.
- **Hybrid Systems.** *Some data stores are loosely coupled, while others are tightly coupled*. Examples: Spark SQL, CloudMdsQL, and BigDAWG.

2.3 Frameworks for Multidatabase Federation Characterization

Besides the taxonomies, Tan *et al.* and Bondiombouy and Valduriez proposed frameworks for federated data systems characterization.

Tan *et al.* proposed a framework inspired on (Sheth and Larson, 1990) and composed by five dimensions: (i) Heterogeneity; (ii) Autonomy; (iii) Transparency; (iv) Flexibility; (v) Optimality.

Heterogeneity: in data-integration systems, implies the design intent is threefold: (i) Uniform access and management of stores' data; (ii) Advantage of component processing engines; (iii) Minimal loss of ex-

pressiveness of the underlying query interfaces. Heterogeneity may be classified in:

1. *Data-store Heterogeneity*: different modeling techniques, *e.g.*, column store, key-value stores.
2. *Processing-engine Heterogeneity*: different processing capabilities, *e.g.*, processing engines modeled around relations, arrays, and graphs.
3. *Query-interface Heterogeneity*: different data models in query engines, supporting various formal algebras, towards expressiveness in a semantic context, *e.g.*, a system with relational and array query interface allows expressing simple linear-algebra operations on relational data.

Autonomy: relates to regulations and constraints:

1. *Association Autonomy*: local data stores decide when to associate and dissociate itself from the federation.
2. *Execution Autonomy*: local data stores may support federation and native applications. They decide prioritization when required.
3. *Evolution Autonomy*: local stores' databases evolve independently from the federation, which has to adapt to the changes.

Transparency: concerns to integration details of storage and layout:

1. *Location Transparency*: local stores provide mechanisms to hide location details.
2. *Transformation Transparency*: local stores provide mechanisms to hide details of data types and structures. Users can focus on logical-level transformations, and the transparent system infers and map data types, and adjust data structures.

Flexibility: concerns the capacity to manage arbitrary formats and to support flexible workflows and user-defined functions.

1. *Schema Flexibility*: allows user-defined schemata and dynamic schema discovery, making it possible to automate transformations.
2. *Interface Flexibility*: the query interface allows user-defined functions and extensions instead of being around a fixed algebra.
3. *Architectural Flexibility*: provide a modularized architecture allowing customization to different scenarios, making possible extending query interface, query optimizer, backend engines, etc.

Optimality: concerns opportunities for optimization through improvements in data placement and federated query plan generation.

1. *Federated Plan Optimization*: federated query plans for sub-queries or data transformation and migration to achieve better performance.
2. *Data-placement Optimization*: place the data in the best fitting engine using, *e.g.*, rule-based and cost-based methods.

Tan *et al.* analyze polystore systems whose design and implementation emphasize query-processing and query-answering challenges, such as BigDAWG, CloudMdsQL, Myria, and Apache Drill.

The framework proposed by Bondiombouy and Valduriez to compare multistore systems is based on two dimensions: (i) *Functionality*: concerns the dimensions objective, data model, query language, and data stores that are supported; (ii) *Implementation Techniques*: concern the dimensions special modules, schema management, query processing, and query optimization (Bondiombouy and Valduriez, 2016).
Related to **Functionality**, they point out:

- The major **Objective** of multistore is the ability to integrate relational data (stored in RDBMS) with other kinds of data in different data stores;
- Each multistore supports different kinds of **Data Stores** (*e.g.*, RDBMS, NoSQL, BigTable, HDFS, Array DBMS, DSMS).
- In terms of the **Data Model**, most systems provide a relational abstraction. BigIntegrator, Polybase, and HadoopDB have relational data models. Forward and CloudMdsQL are JSON-based. QoX has a more general graph abstraction to capture analytic data flows. SparkSQL has a nested model. BigDAWG and Estocada have no unique model since they allow access data stores with their native (or island) languages.
- Considering **Query Language**, most systems provide a SQL-like language, like BigIntegrator, HadoopDB (HiveQL), SparkSQL, CloudMdsQL (with native subqueries). Polybase uses SQL. QoX is XML-Based. Estocada and BigDAWG use native query languages.

Related to **Implementation Techniques:**

- **Special Modules:** refine the generic architecture or bring new functionalities. Examples for the first are importer, absorber and finalizer, query processor, query planner. Examples for the second are dataflow engine, HDFS bridge, storage advisor.
- For **Schema Management**, most multistore manage a Global-as-View (GAV) or a Local-as-view (LAV) [5] global schema approach, which indicates how the elements of the global schema can be derived, when needed, from the elements of the data

[5] In LAV, each data source schema is treated as a view definition of the global schema. In GAV, the global schema is defined as a set of views over the data source schemes.

source schemes (Lenzerini, 2002). *E.g.*, QoX, Estocada, SparkSQL, and CloudMdsQL do not support global schemes, although they provide mechanisms to deal with the data stores' local schemes.

- The **Query Processing** techniques usually are extensions of known techniques from distributed database systems, *e.g.*, data/function shipping, query decomposition (based on the data stores' capabilities, bind join, select pushdown).
- The **Query Optimizations** are usually supported by a (simple) cost model or heuristics.

3 TOOLS

This section characterizes existing polystore tools.

- `BigDAWG` (Big Data Analytics Working Group) (Elmore et al., 2015; Tan et al., 2017)**.**
  - **Description:** Polystore system for large-scale analytics, real-time streaming support, smaller analytics at interactive speeds, data visualization, and query processing over multiple databases. Each storage engine may have a different data model.
  - **Owner/License:** Intel Science and Technology Center for Big Data (ISTC)[6] – BSD-3[7].
  - **Goal:** data integration with federated architecture over collections of vertically integrated database engines.
  - **Internal Data Representation and Platform for Data Operations:** it employs no internal data model or intermediate algebra for query translation and data transformation.
  - **Context Segregation**: BigDAWG separates data in islands where each island has a data model, logical structure, query language or algebra, and one or more backend engines for data storage and query execution, *e.g.*, a relational island may be composed by PostgreSQL and MySQL DBMSs.
  - **Queries Specification:** in the scope of the island, *e.g.*, a SQL query to a relational island.
  - **Query Execution:** queries are decomposed in subqueries executed by the database engines connected to the island. The queries over more than one island are expressed using a SCOPE operator. A CAST operator is used to change the semantic context in a cross-island query.
  - **Heterogeneity:** handled by wrappers' (`shims`).
  - **Main Components:**
    - Query-planning module (planner/optimizer);
    - Performance-monitoring module (monitor);
    - Data-migration module (migrator): moves data across database engines when needed;
    - Query-execution module (executor).
  - **Demonstration:** BigDAWG was demonstrated using MIMIC II (or "Multiparameter Intelligent Monitoring in Intensive Care II") use case, which includes the relational, array, stream, and key-value databases.
- `CloudMdsQL` (Tan et al., 2017; Kolev et al., 2016a; Kolev et al., 2016b)**.**
  - **Definition:** CloudMdsQL (Cloud Multidata store Query Language) is a functional SQL-like language, designed for querying multiple heterogeneous databases within a single query containing nested subqueries. The CloudMdsQL technology is now at LeanXcale, in a proprietary product. Only the compiler remains available for research.
  - **Owner/License:** LeanXcale[8] – proprietary.
  - **Goal**: develop a functional SQL-like language for heterogeneous databases within a single query containing nested subqueries.
  - **Highlight:** CloudMdsQL exploits local data stores by allowing part of the query to be expressed and processed using local native queries (*e.g.*, a breadth-first search in a graph database) which can be called as functions, and at the same time be optimized using a cost model like pushing down select predicates, using binding join, performing join order, or planning intermediate data shipping.
  - **Internal Data Representation and Platform for Data Operations:** it uses a table-based common data model that supports other data types, like arrays and JSON objects to handle non-flat and nested data with basic operators over them.
  - **Context Segregation:** by database engines.
  - **Query Specification:** SQL based on embedded functional subqueries written in the native query languages of the database engines. The language also addresses distributed processing frameworks (*e.g.*, Apache Spark), allowing usage of user-defined map/filter/reduce operator as subqueries.
  - **Query Execution:** a subquery is defined as a named table expression where the user defines

[6]https://bigdawg.mit.edu

[7]https://github.com/bigdawg-istc/bigdawg/blob/master/license.txt

[8]https://www.leanxcale.com

the columns and types of the table and expression in SQL SELECT (which the compiler can analyze and possibly rewrite) or a native expression (which is directly delegated to the corresponding data store). *E.g.*, two named table expressions may query a relational database and a document store, and be joined to produce the result. The query compiler decomposes the query into a query execution plan (QEP) in a directed acyclic graph of relational operators. Leaf nodes correspond to subqueries to be executed by the wrappers over the data stores.

- **Heterogeneity:** is handled through a mediator/wrapper architecture and the table-based common model.
- **Main Components:**
  - Query Planner: performs lexical and syntax analysis besides query rewrite plans;
  - Capability Manager: validates rewritten subqueries against datastore capabilities;
  - Query Optimizer: uses cost functions and database statistics in a cost model to select the best plan. Besides, users may define cost and selectivity functions. The optimizer may rewrite the QEP generated by the query compiler. CloudMdsQL uses bind join to perform semi-joins across heterogeneous data stores.
  - QEP Builder: generates plans and serializes them in JSON.
  - Query Execution Controller: parses QEP, identifies sub-plans, and invokes wrappers.
  - Finalizer: translates subquery plans into native queries that can be executed by the engine.
  - X-Ray (Guimarães and Pereira, 2015): Monitors execution. .

- `Myria`[9] (Tan et al., 2017; Halperin et al., 2014)**.**
  - **Definition:** cloud service for big data management and analytics.
  - **Goal:** simplify data upload and data science tasks with efficient query execution to process and explore the data.
  - **Owner/License:** the University of Washington[10] – BSD 3[11].
  - **Internal Data Representation and Platform for Data Operations:** relational data representation with a relational-algebra compiler.
  - **Context Segregation:** in the level of the encapsulated data stores.
  - **Query Specification:** in MyriaL, an imperative-declarative hybrid language, or via a Python API. It supports user-defined function (UDP) and user-defined aggregates (UDA) via an exposed Python API.
  - **Query Execution:** the Relational Algebra Compiler (RACO) parses queries in Myria algebra and transforms them into the specific API calls or the query primitives supported by the local database engines. RACO uses rule-based optimization to generate federated query plans that take advantage of the performance characteristics of the supported database engines.
  - **Main Components:**
    - MariaX: query-execution engine that uses a parallel, pipelined, possibly cyclic graph of dataflow operators with built-in support for asynchronous evaluation of recursive queries.
    - RACO: query optimizer and federated query executor that uses relational algebra extended with imperative constructs capturing the semantics of array, graph, and key-value data models.

- `Apache Drill` (Tan et al., 2017; Hausenblas and Nadeau, 2013)**.**
  - **Definition:** distributed, massively parallel query engine.
  - **Owner/License:** the Apache Software Foundation[12] – Apache License[13].
  - **Goal:** answer fast to *ad-hoc* queries over a huge amount of unstructured and weakly structured data spread across servers.
  - **Internal Data Representation and Platform for Data Operations:** JSON-based data model.
  - **Context Segregation:** in the level of the encapsulated data stores.
  - **Query Specification:** supports ANSI SQL and MongoDB QL, and user-defined functions.
  - **Query Execution:** queries are parsed and transformed into a logical plan, which is transformed and optimized into a physical plan.
  - **Main Components:**
    - Drillbit: daemon service running cluster nodes aiming at maximizing data locality.
    - Zookeeper: broker between the clients and Drillbits, and among Drillbits.

[9] https://myria.cs.washington.edu/
[10] http://www.washington.edu/
[11] https://github.com/uwescience/myria/blob/master/LICENSE
[12] https://drill.apache.org/
[13] https://drill.apache.org/apacheASF/

4 REQUIREMENTS AND RESEARCH CHALLENGES

The modern federated database system main requirements are:

- **Location and Data Sources Encapsulation** (Elmore et al., 2015)**:** Provide a smooth interface to free programmers from having to learn several query languages and store engines.
- The **Deployment:** should support cloud practices (Halperin et al., 2014) complexity and cost of installation, administration, and maintenance should not be prohibitive; mechanisms for predicting and debugging performance, as well as controlling costs, should be available.
- **Query Language** (Kolev et al., 2016b)**:** should work on heterogeneous data stores; support arbitrary chain of queries, *i.e.*, a query result in one database be input for a query in another database; be schema independent, *i.e.*, allow the integration of databases with or without schema; allow data-metadata transformation, *e.g.*, convert attributes or relations into data and vice-versa.
- **Query Tools:** should support users and algorithm designers with an easy-to-use set of interfaces, languages, and APIs that scale from simple SPJ (Select-Project-Join) queries to advanced application-specific ones (Halperin et al., 2014).
- **Processing** (Halperin et al., 2014)**:** should be efficient and support: scale queries and optimization combining state-of-the-art and novel techniques, *e.g.*, use of bind joins, semi-joins, core parallel query processing concepts.
- **Real-time Decision Support** (Elmore et al., 2015)**:** through stream processing able to connect historical and stream data.
- **Visualization Tools** (Elmore et al., 2015)**:** supporting disparate data models and new user interaction mechanisms. *E.g.*, big data application visualization should support questions like "give me something interesting from data" in an exploratory way.
- **Shuffle Data among Backends** (Elmore et al., 2015)**:** *i.e.*, supporting moving data and intermediate results from one storage to another as needed to fit a user's query and high performance. *E.g.*, each engine may know how to read binary data in parallel directly from another engine (Elmore et al., 2015). Using a monitoring system that learns types of queries, and move data accordingly, *i.e.*, it transfers the data to the engine that has the best data model to answer the query.
- **Cross-system Solution** (Elmore et al., 2015)**:** able to include other polystores.

The research challenges are (Stonebraker, 2015)(Bondiombouy and Valduriez, 2016)(Elmore et al., 2015):

- **Query Language:** *easy to use query language with efficient processing over diverse stores.* SQL-like language facilitates integration with existing tools but comprises efficiency. Alternatives are to access stores directly or to use a functional query language that allows native subqueries as functions within the query language.
- **Complex Analytics:** *efficient combination of data from multiple stores with linear algebra algorithms.* Analytics tasks have been moved from relational analytics (*e.g.*, COUNT, SUM, AVG with group by) to predictive models (*e.g.*, machine learning, regression, statics). The majority of such predictive models are based on linear algebra algorithms, such as regression analysis, singular value decomposition, eigenanalysis, k-means clustering, etc. Although linear algebra packages are optimized by software and hardware, they have different characteristics when compared to data systems, *e.g.*, size of computation tiles, choice of networks, compression techniques. So, algebra packages and data stores should be decoupled, and data should be converted back and forth between them in a non-expensive manner.
- **Query Optimization:** *optimization should handle data-flow and multi-platform scheduling* able to update the cost model or add new heuristics when data stores join or disjoin the system.
- **Semantic Mapping and Record Linkage:** *automatic translation of utterances to the local dialect of a storage system and integration of the results*.
- **Automatic Copy:** *efficient copy of data between stores and federated system*, considering data transformation and memory access techniques.
- **Distributed Transactions:** *handle transactions over distributed, heterogeneous store systems with diverse local transaction models.* The problem is harder when considering, *e.g.*, NoSQL data stores that do not provide ACID[14] transaction support.
- **Automatic Load Balancing and Provisioning:** *automatically balance data load among data stores* by replications or moving data, which requires a monitoring feature.
- **Novel User Interfaces:** *innovative data mining, visualization, and browsing user interfaces.* Typical user interaction flows may not fit modern fed-

[14] Atomicity, Consistency, Isolation, and Durability.

eration. *E.g.*, as new data sources may join or disjoin the federation, new data may rise or disappear, which brings new knowledge.

- **Benchmarks:** *should be developed to evaluate federations* addressing store combinations and multiple query language processing, like PolyBench (Karimov et al., 2018).

5 CONCLUSION

Modern applications require the manipulation of structured and unstructured data, usually in high volume, over distributed and heterogeneous data sources. The pattern "one size fits all" does not hold anymore. Thus, innovative solutions capable of accessing and manipulating data in such an environment are required.

This work presented the state-of-the-art, detailed solutions, their main components, how queries are specified and executed, and other features. Afterward, we presented guidelines and challenges the solutions should address. Researchers and practitioners can use our finds to focus their work.

As future work, we aim at evaluating the tools in practice through case studies and experiments to identify the level they meet the challenges and to bring new open issues. We also intend to perform a systematic review towards a broader analysis improving the overview presented in this work.

REFERENCES

Angles, R. and Gutierrez, C. (2008). Survey of graph database models. *ACM Comp. Surveys*, 40(1):1–39.

Bondiombouy, C. and Valduriez, P. (2016). Query processing in multistore systems: an overview. Research Report RR-8890, INRIA.

Elmore, A., Duggan, J., Stonebraker, M., Balazinska, M., Cetintemel, U., Gadepally, V., Heer, J., Howe, B., Kepner, J., Kraska, T., et al. (2015). A demonstration of the bigdawg polystore system. *Proceedings of the VLDB Endowment*, 8(12):1908–1911.

Guimarães, P. and Pereira, J. (2015). X-ray: Monitoring and analysis of distributed database queries. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 80–93. Springer.

Gurusamy, V., Kannan, S., and Nandhini, K. (2017). The Real Time Big Data Processing Framework: Advantages and Limitations. *Intl. Journal of Computer Sciences and Engineering (JCSE)*, 5(12):305–312.

Halperin, D., Teixeira de Almeida, V., Choo, L. L., and et al. (2014). Demonstration of the myria big data management service. In *Proceedings of the 2014 ACM SIGMOD Intl. Conf. on Mngt. of Data*, pages 881–884.

Haslhofer, B., Momeni Roochi, E., Schandl, B., and Zander, S. (2011). Europeana rdf store report. Technical report, University of Vienna.

Hausenblas, M. and Nadeau, J. (2013). Apache drill: interactive ad-hoc analysis at scale. *Big data*, 1(2):100–104.

Iancu, B. and Georgescu, T. M. (2018). Saving Large Semantic Data in Cloud: A Survey of the Main DBaaS Solutions. *Informatica Economica*, 22(1).

Karimov, J., Rabl, T., and Markl, V. (2018). Polybench: The first benchmark for polystores. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 24–41. Springer.

Kolev, B., Bondiombouy, C., Valduriez, P., Jiménez-Peris, R., Pau, R., and Pereira, J. (2016a). The CloudMdsQL Multistore System. In *Proc. of Intl. Conf. on Management of Data (SIGMOD'16)*, pages 2113–2116. ACM.

Kolev, B., Valduriez, P., Bondiombouy, C., Jimenez-Peris, R., Pau, R., and Pereira, J. (2016b). CloudMdsQL: Querying Heterogeneous Cloud Data Stores with a Common Language. *Distributed and parallel databases*, 34(4):463–503.

Lenzerini, M. (2002). Data integration: A theoretical perspective. In *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 233–246. ACM.

Modoni, G. E., Sacco, M., and Terkaj, W. (2014). A survey of rdf store solutions. In *Intl. Conf. on Engineering, Technology and Innovation (ICE)*, pages 1–7. IEEE.

Nayak, A., Poriya, A., and Poojary, D. (2013). Type of NOSQL databases and its comparison with relational databases. *International Journal of Applied Information Systems*, 5(4):16–19.

Özsu, M. T. and Valduriez, P. (2020). *Principles of distributed database systems*. Springer, 4th edition.

Sheth, A. P. and Larson, J. A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys (CSUR)*, 22(3):183–236.

Stonebraker, M. (2015). The case for polystore. https://wp.sigmod.org/?p=1629.

Stonebraker, M., Bear, C., Çetintemel, U., Cherniack, M., Ge, T., Hachem, N., Harizopoulos, S., Lifter, J., Rogers, J., and Zdonik, S. (2007). One size fits all? part 2: Benchmarking results. In *Proc. CIDR*.

Tan, R., Chirkova, R., Gadepally, V., and Mattson, T. G. (2017). Enabling query processing across heterogeneous data models: A survey. In *IEEE Intl. Conf. on Big Data (Big Data)*, pages 3211–3220. IEEE.

Zheng, Z., Wang, P., Liu, J., and Sun, S. (2015). Real-Time Big Data Processing Framework: Challenges and Solutions. *Applied Math. & Inf. Sciences*, 9(6):3169.

Zulkefli, N. S. S., Rahman, N. A., Bakar, Z. A., Nordin, S., Sembok, T. M. T., and Teo, N. H. I. (2013). Evaluation of triple indices in retrieving web documents. In *Intl. Conf. on Advanced Computer Science Applications and Technologies (ACSAT)*, pages 525–529.