Chapter 5
Federated Query Processing

Kemele M. Endris[1,2], Maria-Esther Vidal[1], and Damien Graux[3]

[1] TIB Leibniz Information Centre For Science and Technology, Hannover, Germany
[2] L3S Research Center, Hannover, Germany
[3] ADAPT SFI Research Centre, Trinity College Dublin, Ireland

Abstract. Big data plays a relevant role in promoting both manufacturing and scientific development through industrial digitization and emerging interdisciplinary research. Semantic web technologies have also experienced great progress, and scientific communities and practitioners have contributed to the problem of big data management with ontological models, controlled vocabularies, linked datasets, data models, query languages, as well as tools for transforming big data into knowledge from which decisions can be made. Despite the significant impact of big data and semantic web technologies, we are entering into a new era where domains like genomics are projected to grow very rapidly in the next decade. In this next era, integrating big data demands novel and scalable tools for enabling not only big data ingestion and curation but also efficient large-scale exploration and discovery. Federated query processing techniques provide a solution to scale up to large volumes of data distributed across multiple data sources. Federated query processing techniques resort to source descriptions to identify relevant data sources for a query, as well as to find efficient execution plans that minimize the total execution time of a query and maximize the completeness of the answers. This chapter summarizes the main characteristics of a federated query engine, reviews the current state of the field, and outlines the problems that still remain open and represent grand challenges for the area.

1 Introduction

The number and variety of data collections have grown exponentially over recent decades and a similar growth rate is expected in the coming years. In order to transform the enormous amount of disparate data into knowledge from where actions can be taken, fundamental problems, such as data integration and query processing, must be solved. Data integration requires the effective identification of entities that, albeit described differently, correspond to the same real-world entity. Moreover, data is usually ingested in myriad unstructured formats and may suffer reduced quality due to biases, ambiguities, and noise. These issues impact on the complexity of the solutions for data integration. Semantic integration of big data entails variety by enabling the resolution of several interoperability conflicts [159, 444], e.g., structuredness, schematic, representation, completeness,

granularity, and entity matching conflicts. Conflicts arise because data sources may have different data models or none, follow various schemes for data representation, and contain complementary information. Furthermore, a real-world entity may be represented using diverse properties or at various levels of detail. Thus, techniques able to solve interoperability issues while addressing data complexity challenges imposed by big data characteristics are required [400].

Existing solutions to the problem of query processing over heterogeneous datasets rely on a unified interface for overcoming interoperability issues, usually based on metamodels [223]. Different approaches have been proposed, mainly with a focus on data ingestion and metadata extraction and management. Exemplary approaches include GEMMS [363], PolyWeb [243], BigDAWG [118], Ontario [125], and Constance [179]. These systems collect metadata about the main characteristics of the heterogeneous data collections, e.g., formats and query capabilities. Additionally, they resort to a global ontology to describe contextual information and relationships among data sets. Rich descriptions of the properties and capabilities of the data have shown to be crucial for enabling these systems to effectively perform query processing.

In the context of the Semantic Web, the problem of federated query processing has also been actively studied. As a result, diverse federated SPARQL query engines have been defined that enable users to execute queries over a federation of SPARQL endpoints. State-of-the-art techniques include FedX [387], ANAPSID [6], and MULDER [124]. FedX implements adaptive techniques to identify relevant sources to evaluate a query. It is able to contact SPARQL endpoints on the fly to decide the subqueries of the original query that can be executed over the endpoints of the federation. ANAPSID makes use of metadata about the vocabularies used on the RDF datasets to perform source selection. Based on the selected sources, ANAPSID decomposes original queries and finds efficient plans to collect the answers incrementally. Finally, MULDER resorts to description of the RDF datasets based on the classes and relations of the dataset vocabularies. MULDER proposes the concept of the RDF Molecule Templates (RDF-MTs) to describe the datasets and efficiently perform source selection and query planning. The rich repertoire of federated query engines just reveals the importance of query processing against the RDF dataset, as well as the attention that the problem has received from the database and semantic web communities.

The contributions of the work are summarized as follows:

- A description of the concept of the data integration system and an analysis of the different parameters that impact on the complexity of a system.
- A characterization of the challenges addressed by federated query engines and analysis of the current state of the federated query processing field.
- A discussion of the analysis of the grand challenges in this area and future directions.

The remainder of the chapter is structured as follows: Section 2 presents an overview of the data integration system and the roles that they play in the problem of accessing and processing queries over heterogeneous data sources. Section 3 describes the problem of federated query processing, the main challenges to be

addressed by a federated query engine, and the state of the art. Finally, grand challenges and future directions are outlined in Section 4.

2 Data Integration Systems

An enormous amount of data is being published on the web [377]. In addition, different data sources are being generated and stored within enterprises as well due to technological advances in data collection, generation, and storage. These data sources are created independently of each other and might belong to different administrative entities. Hence, they have different data representation formats as well as access interfaces. Such properties of the data sources hinder the usage of the information available in them. Data integration is the process of providing uniform access to a set of distributed (or decentralised), autonomous, and heterogeneous data sources [113]. Data integration systems provide a global schema (also known as mediated schema) to provide a reconciled view of all data available in different data sources. Mapping between the global schema and source schema should be established to combine data residing in data sources considered in the integration process. Generally, data integration system is formally defined as follows [278]:

Definition 1 (Data Integration System). *A data integration system, $\mathbb{I}$, is defined as a triple $< G, S, M >$, where:*

- *G is the global schema, expressed in a language L_G over an alphabet A_G. The alphabet comprises a symbol for each element of G.*
- *S is the source schema, expressed in a language L_S over an alphabet A_S. The alphabet A_S includes a symbol for each element of the sources.*
- *M is the mapping between G and S, constituted by a set of assertions of the forms: $q_S \rightarrow q_G$, $q_G \rightarrow q_S$; where q_S and q_G are two queries of the same arity, respectively over the source schema S, and over the global schema G. An assertion specifies the connection between the elements of the global schema and those of the source schema.*

Defining schema mapping is one of the main tasks in a data integration system. Schema mapping is the specification of correspondences between the data at the sources and the global schema. The mappings determine how the queries posed by the user using the global schema are answered by translating to the schema of the source that stores the data. Two basic approaches for specifying such mappings have been proposed in the literature for data integration systems are *Global-as-View(GAV) [139, 180]* and *Local-as-View(LAV)* [280, 431].

Rules defined using the Global-as-View (GAV) approach define concepts in the global schema as a set of views over the data sources. Using the GAV approach, the mapping rules in M define the concepts of the schema in the sources, S, with each element in the global schema. A query posed over the global schema, G, needs to be reformulated by rewriting the query with the views defined in, M. Such rewriting is also known as *query unfolding* – the process of rewriting

the query defined over global schema to a query that only refers to the source schema. Conceptually, GAV mappings specify directly how to compute tuples of the global schema relations from tuples in the sources. This characteristics of GAV mappings makes them easier for query unfolding strategy. However, adding and removing sources in the GAV approach may involve updating all the mappings in the global schema, which requires knowledge of all the sources. Mappings specified using the Local-as-View (LAV) approach describe the data sources as views over the global schema, contrary to the GAV approach that defines the global schema as views over the data sources. Using the LAV approach, the mapping rules in M associates a query defined over the global schema, G, to each elements of source schema, S. Adding and removing sources in LAV is easier than GAV, as data sources are described independently of each other. In addition, it allows for expressing incomplete information as the global schema represents a database whose tuples are unknown, i.e., the mapping M defined by LAV approach might not contain all the corresponding sources for all the elements in the global schema, G. As a result, query answering in LAV may consist of querying incomplete information, which is computationally more expensive [113].

In this chapter, we define a source description model, RDF Molecule Template (RDF-MT), an abstract description of entities that share the same characteristics, based on the GAV approach. The global schema is defined as a consolidation of RDF-MTs extracted from each data source in the federation. Rule-based mappings, such as RML, are used to define the GAV mappings of heterogeneous data sources. RDF-MTs are merged based on their semantic descriptions defined by the ontology, e.g., in RDFS.

2.1 Classification of Data Integration Systems

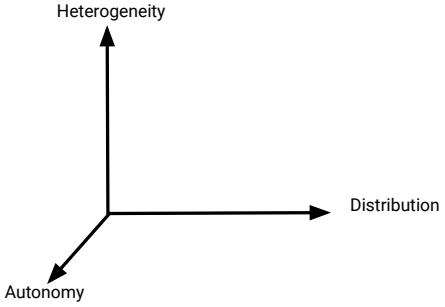

Fig. 1: Dimensions of Data Integration Systems

Data integration systems can be classified with respect to the following three dimensions: autonomy, distribution, and heterogeneity [336], Figure 1. *Autonomy* dimension characterizes the degree to which the integration system allows

each data source in the integration to operate independently. Data sources have autonomy over choice of their data model, schema, and evolution. Furthermore, sources also have autonomy to join or leave the integration system at any time as well as to select which fragments of data to be accessible by the integration system and its users. *Distribution* dimension specifies the data that is physically distributed across computer networks. Such distribution (or decentralization) can be achieved by controlled distribution or by the autonomous decision of the data providers. Finally, *heterogeneity* may occur due to the fact that autonomous development of systems yields different solutions, for reasons such as different understanding and modeling of the same real-world concepts, the technical environment, and particular requirements of the application [336]. Though there are different types of heterogeneity of data sources, the important ones with respect to data interoperability are related to data model, semantic, and interface heterogeneity. Data model heterogeneity captures the heterogeneity created by various modeling techniques such that each data model has different expressive power and limitations, e.g., relational tables, property graph, and RDF. Semantic heterogeneity concerns the semantics of data and schema in each source. The semantics of the data stored in each source are defined through the explicit definition of their meanings in the schema element. Finally, interface heterogeneity exists if data sources in the integration system are accessible via different query languages, e.g., SQL, Cypher, SPARQL, and API call.

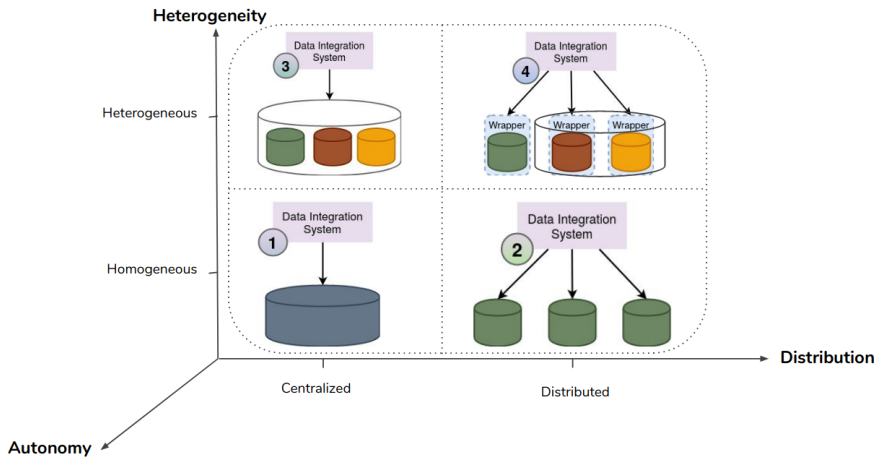

Fig. 2: Classification of Data Integration Systems

Figure 2 shows different classifications of data integration systems with respect to distribution and heterogeneity dimensions. The first type of data integration systems, Figure 2.(1), loads heterogeneous data from data sources to a centralized storage after transforming them to a common data representation format. The second type of data integration systems, Figure 2.(2), supports data

distributed across networks; however, they only support if the data sources in the system are homogeneous in terms of data model and access methods. The third type of data integration systems, Figure 2.(3), supports data heterogeneity among data sources in the integration system. However, these data integration systems are managed in a centralized way and data is stored in a distributed file system (DFS), such as Hadoop [4]. Finally, the fourth type of data integration systems, Figure 2.(4), supports data distributed across networks as well as heterogeneity of data sources. Such integration systems utilize special software components to extract data from the data sources using native query language and access mechanism. They can also transform data extracted from the sources to data representation defined by the integration system. Data sources in the integration system might also be autonomous. Such types of system are different from the third type by how data is distributed and stored. While the fourth type supports any storage management, including DFS, the third type of data integration systems supports only DFS in a centralized way. Mostly the distribution task is handled by the file system. For instance, data might be stored in a multi-modal data management system or in Data Lake storage based only on a distributed file system (DFS). In the third type of data integration system, data is loaded from the original source to the centralized storage for further processing. Federated query processing systems fall in the second and fourth type of integration system when the data sources are autonomous.

Data integration systems also have to make sure that data that is current (fresh) is accessed and integrated. Especially, for DFS-based Data Lakes, Figure 2.(2), and the centralized, Figure 2.(4), integration systems, updates of the original data sources should be propagated to guarantee the freshness of data. Furthermore, when accessing an original data source from the provider is restricted, or management of data in a local replica is preferred, integration systems Figure 2.(1) and (3), need to guarantee data freshness by propagating changes.

2.2 Data Integration in the era of Big Data

In the era of big data, a large amount of structured, semi-structured, and unstructured data is being generated at a faster rate than ever before. Big data systems that integrate different data sources need to handle such characteristics of data efficiently and effectively. Generally, big data is defined as data whose volume, acquisition speed, data representation, veracity, and potential value overcome the capacity of traditional data management systems [76]. Big data is characterized by the 5Vs model: *Volume* denotes that generation and collection of data are produced at increasingly big scales. *Velocity* represents that data is generated and collected rapidly. *Variety* indicates heterogeneity in data types, formats, structuredness, and data generation scale. Veracity refers to noise and quality issues in the data. Finally, *Value* denotes the benefit and usefulness that can be obtained from processing and mining big data.

[4] https://hadoop.apache.org/

There are two data access strategies for data integration: *schema-on-write* and *schema-on-read*. In the schema-on-write strategy, data is cleansed, organized, and transformed according to a pre-defined schema before loading to the repository. In schema-on-read strategy, raw data is loaded to the repository as-is and schema is defined only when the data is needed for processing [26]. Data warehouses provide a common schema and require data cleansing, aggregation, and transformation in advance, hence, following the schema-on-write strategy. To provide scalable and flexible data discovery, analysis, and reporting, *Data Lakes* have been proposed. Unlike data warehouses, where data is loaded to the repository after it is transformed to a target schema and data representation, Data Lakes store data in its original format, i.e., the schema-on-read strategy. Data Lakes provide a central repository for raw data that is made available to the user immediately and defer any aggregation or transformation tasks to the data analysis phase, thus addressing the problem of disconnected information silos, which is the result of non-integrated heterogeneous data sources in isolated repositories with diverse schema and query languages. Such a central repository may include different data management systems, such as distributed file systems, relational database management systems, graph data management systems, as well as triple stores for specialized data model and storage.

3 Federated Query Processing

A federated query processing system[5], provides a unified access interface to a set of autonomous, distributed, and heterogeneous data sources. While distributed query processing systems have control over each dataset, federated query processing engines have no control over datasets in the federation and data providers can join or leave the federation at any time and modify their datasets independently. Query processing in the context of data sources in a federation is more difficult than in centralized systems because of the different parameters involved that affect the performance of the query processing engine [113]. Data sources in a federation might contain fragments of data about an entity, have different processing capabilities, support different access patterns, access methods, and operators. The role of a federated query engine is to transform a query expressed in terms of the global schema, i.e., the federated query, into an equivalent query expressed in the schema of the data sources, i.e., local query. The local query represents the actual execution plan of the federated query by the data sources of the federation. The transformation of the federated query to a local query needs to be both effective and efficient. Query transformations are effective if the generated query is equivalent to the original one, i.e., both the original and the transformed queries produce same results. On the other hand, query transformations are efficient if the execution strategy of the transformed query makes use of minimum computational resources and communication cost. Producing an efficient execution strategy is difficult as many equivalent and correct

[5] We use the terms *federated query processing system*, *federated query engine*, and *federated query processing system* interchangeably.

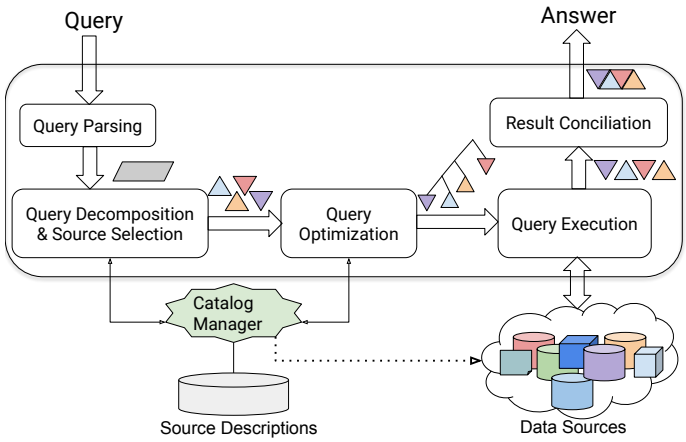

Fig. 3: Federated Query Processing Basic Components

transformations can be produced and each equivalent execution strategy leads to different consumption of resources [336]. The main objective of federated query processing is to transform a query posed on a federation of data sources into a query composed of the union of subqueries over individual data sources of the federation. Further, a query plan is generated in order to speed up the processing of each individual subquery over the selected sources, as well as the gathering of the results into the query answer. An important part of query processing in the context of federated data sources is query optimization as many execution plans are correct transformations of the same federated query. The one that optimizes (minimizes) resource consumption should be retained. Query processing performance can be measured by the total cost that will be used in query processing and the response time of the query, i.e., time elapsed for executing the query.

As an RDF data model continues gaining popularity, publicly available RDF datasets are growing in number and size. One of the challenges emerging from this trend is how to efficiently and effectively execute queries over a set of autonomous RDF datasets. Saleem et al. [378] study federated RDF query engines with web access interfaces. Based on their survey results, the authors divide federation approaches into three main categories: *Query Federation over SPARQL endpoints*, *Query Federation over Linked Data (via URI lookups)*, and *Query Federation on top of Distributed Hash Tables*. Moreover, Acosta et al. [5] classified federated RDF query processing engines based on the type of data sources they support into three categories: *Federation of SPARQL endpoints*, *Federation of RDF Documents*, and *Federation of Triple Pattern Fragments*.

Conceptually, federated query processing involves four main sub-problems (components): (i) *data source description*, (ii) *query decomposition and source selection*, (iii) *query planning and optimization*, and (iv) *query execution*. Federated query engines also include two additional sub-problems: *query parsing* and *result conciliation*. Query parsing and result conciliation sub-problems deal with

syntactic issues of the given query and formatting the results returned from the query execution, respectively. Below we provide an overview of the data source description, query decomposition and source selection, query planning and optimization as well as query execution sub-problems.

3.1 Data Source Description

The data source description sub-problem deals with describing the data available in data sources and managing catalogs about data sources that are participating in the federation. Data source descriptions encode information about available data sources in the federation, types of data in each data source, access method of data sources, and privacy and access policies of these data sources [113]. The specification of what data exist in data sources and how the terms used in data sources are related to the global schema are specified by the schema mapping. Schema mappings also represent privacy and access control restrictions as well as statistics on the available data in each data source. Federated query engines rely on the description of data sources in the federation to select relevant sources that may contribute to answer a query. Data source descriptions are utilized by source selection, query decomposition, and query optimization sub-problems.

A catalog of data source descriptions can be collected offline or during query running-time. Based on the employed catalog of source descriptions, SPARQL federation approaches can be divided into three categories [378]: *pre-computed catalog assisted, on-the-fly catalog assisted, and hybrid (uses both pre-computed and on-the-fly)* solutions. Pre-computed catalog-assisted federated SPARQL query engines use three types of catalogs: service descriptions, VoID (*Vocabulary of Interlinked Datasets*) description, and list of predicates [333]. The first two catalogs are computed and published by the data source providers that contains descriptions about the set of vocabularies used, a list of classes and predicates, as well as some statistics about the instances such as number of triples per predicate, or class. Specifically in VoID descriptions, there is information about external linksets that indicate the existence of *owl:sameAs* and other linking properties. The third type of catalog, i.e., a list of predicates, is generated by contacting the data source endpoints and issuing SPARQL queries and extracting predicates from the other two types of catalog.

FedX [387] does not require a catalog of source descriptions computed beforehand but uses triple pattern-wise ASK queries sent to data sources at query time. Triple pattern-wise ASK queries are SPARQL ASK queries which contain only one triple pattern in the graph expression of the given query. Lusail [4], like FedX, uses an on-the-fly catalog solution for source selection and decomposition. Unlike FedX, Lusail takes an additional step to check if pairs of triple patterns can be evaluated as one subquery over a specific endpoint; this knowledge is exploited by Lusail during query decomposition and optimization. Posting too many SPARQL ASK queries can be a burden for data sources that have limited compute resources, which may result in DoS. Pre-computed catalog of data source descriptions can be used to reduce the number of requests sent to the data sources. ANAPSID [6] is a federated query processing engine that employs a hy-

brid solution and collects a list of RDF predicates of the triple patterns that can be answered by the data sources and sends ASK queries when required during query time. Publicly available dataset metadata are utilized by some federated query processing engines as catalogs of source descriptions. SPLENDID [160] relies on instance-level metadata available as *Vocabulary of Interlinked Datasets* (VoID) [10] for describing the sources in a federation. SPLENDID provides a hybrid solution by combining VoID descriptions for data source selection along with SPARQL ASK queries submitted to each dataset at run-time for verification. Statistical information for each predicate and types in the dataset are organized as inverted indices, which will be used for data source selection and join order optimization. Similarly, Semagrow [74] implements a hybrid solution, like SPLENDID, and triple pattern-wise source selection method which uses VoID descriptions (if available) and SPARQL ASK queries.

MULDER [124] and Ontario [125] federated query engine employs source description computed based on the concept of RDF molecules; a set of triples that share the same subject values are called *RDF Molecules*. RDF Molecule Templates (RDF-MTs) encode an abstract description of a set of RDF molecules that share similar characteristics such as semantic type of entities. RDF Molecule Template-based source descriptions leverage the semantics encoded in data sources. It is composed of a semantic concept shared by RDF molecules, a set of mapping rules, a list of properties that a molecule can have, and a list of intra- and inter-connections between other RDF molecule templates. Such description models provide a holistic view over the set of entities and their relationships within the data sources in the federation. For instance, Figure 4 shows RDF-MT based descriptions of the FedBench benchmark composed on 10 RDF data sources.

3.2 Query Decomposition and Source Selection

Selecting the relevant data sources for a given query is one of the sub-problems in federated query processing. Given a federated query parsed with no syntactic problems, the query is first checked if it is semantically correct with respect to the global schema. This step eliminates an incorrect query that yields no results early on. The query is then simplified by, for example, removing redundant predicates. The task of source selection is to select the actual implementation of subqueries in the federation at specific data sources. The sources schema and global schema are given by the data source descriptions as input to this sub-problem. The query decomposition and source selection sub-problem decomposes the federated query into subqueries associated with data sources in the federation that are selected for executing the subqueries. The number of data sources considered for selection are bounded by the data source description given to the federated query processing engine. Each sub-query may be associated to zero or more data source, thus, if the query contains at least one sub-query without data source(s) associated with it, then the global query can be rejected. Source selection task is a critical part of query optimization. Failure to select correct data sources might lead to incomplete answers as well as high response time and resource consumption. The output of this component is a decomposed query into subqueries that

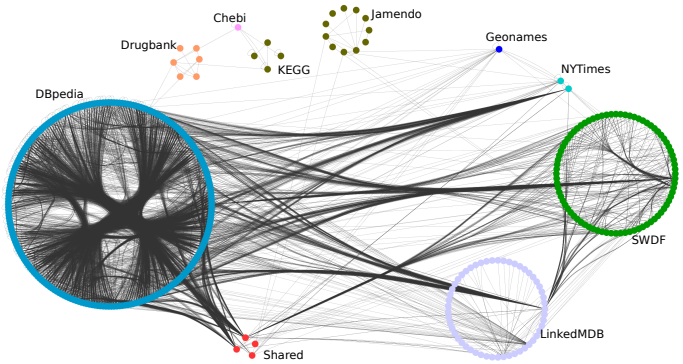

Fig. 4: **RDF-MT-based description of FedBench.** The graph comprises 387 RDF-MTs and 6,317 intra- and inter-dataset links. The dots in each circle represent RDF-MTs. A line between dots in the same circle shows intra-dataset links, while a line between dots in different circles corresponds to inter-dataset links. In numbers, there is only one RDF-MT in ChEBI, 234 in DBpedia, six in Drugbank, one in Geonames, 11 in Jamendo, four in KEGG, 53 in LinkedMDB, two in NYTimes, and 80 in SWDF dataset. Four of these RDF-MTs belong to at least two FedBench datasets, modeled as separate circular dots.

are associated with the selected data sources in the federation. Identifying the relevant sources of a query not only leads to a complete answer but also faster execution time.

3.3 Query Planning and Optimization

The goal of query planning is to generate an execution plan that represent the steps on how the query is executed and which algorithms (operators) are used. The task of query plan generation produces query execution plans, e.g., a tree-based plan where the leaf of the tree corresponds to the sub-queries to be executed in selected data sources and the internal nodes corresponds to the physical (algebraic) operators, such as join, union, project, and filter, that perform algebraic operations by the federated query processing engine. Many semantically equivalent execution plans can be found by permuting the order of operators and subqueries. However, the cost of executing different ordering of a query is not always the same. In a federated setting, the number of intermediate results as well as the communication costs impacts the performance of query execution. Federated query processing engines should use an optimization techniques to select an optimal execution plan that reduces execution time and resource usage, such as memory, communication, etc. Optimization of the query execution plan starts from selecting only relevant sources, decomposition and finally making decisions on the selection of appropriate implementation of join operations. These optimization techniques include making decisions on selection of the join

methods, ordering, and adapting to the condition of the sources. The objective of the planning and optimization sub-problem is to find an execution plan that minimizes the cost of processing the given query, i.e., finding the "best" ordering of operators in the query, which is close to optimal solution. Finding an optimal solution is computationally intractable [209]. Assuming a simplified cost function, it is proven that the minimization of this cost function for a query with many joins is NP-Complete. To select the ordering of operators, it is necessary to estimate execution costs of alternative candidate orderings. There are two type of query optimization in the literature: cost-based and heuristics-based query optimization. In cost-based optimization techniques, estimating the cost of the generated plans, i.e., candidate orderings, requires collecting statistics on each of the data sources either before query executions, *static optimization* or during query execution, *dynamic optimization*. In federated settings, where data sources are autonomous, collecting such statistics might not always be possible. Cost-based approaches are often not possible because the data source descriptions do not have the needed statistics. Heuristic-based optimization techniques can be used to estimate the execution cost using minimum information collected from sources as well as the properties of the operators in the query, such as type of predicates, operators, etc. The output of the query planning and optimization is an optimized query, i.e., query execution plan, with operations (join, union) between subqueries.

3.4 Query Execution

Query execution is performed by data sources that are involved in answering sub-query(s) of the given query. Each sub-query executed in each data source is then optimized using the local schema and index (if available) of the data source and executed. The physical operator (and algorithms) to perform the relational operators (join, union, filter) may be chosen. Five different join methods are used in federated query engines: nested loop join, bound-join, hash join, symmetric join, and multiple join [333]. In nested-loop join (NLJ) the inner sub-query is executed for every binding of the intermediate results from the outer sub-query of the join. The bindings that satisfy the join condition are then included in the join results. Bound-join, like NLJ, executes inner sub-query for the set of bindings, unlike NLJ which executes the inner sub-query for every single binding of the intermediate results from the outer sub-query. This set of bindings can be sent as a UNION or FILTER SPARQL operators can be used to send multiple bindings to the inner sub-query. In the hash-join method, each sub-query (operands of the join operation) is executed in parallel and the join is performed locally using a single hash table at the query engine. The fourth type of join method, symmetric (hash) join, is a non-blocking hash-based join that pipelines parallel execution of the operands and generates output of the join operation as early as possible. Several extended versions of this method are available, such as XJoin [434], agjoin [6], and adjoin [6]. Finally, the multiple (hash) join method uses multiple hash tables to join more than two sub-queries running at the same time.

4 Grand Challenges and Future Work

In this section, we analyze the grand challenges to be addressed in the definition and implementation of federated query engines against distributed sources of big data. These challenges can be summarized as follows:

- Definition of formal models able to describe not only the properties and relationships among data sources, but also represent and explain causality relations, bias, and trustworthiness.
- Adaptive query processing techniques able to adjust query processing schedules according to the availability of the data, as well as to the validity and trustworthiness of the published data.
- Machine learning models able to predict the cost of integrating different sources, and the benefits that the fusion of new data sources adds to the accuracy, validity, and trustworthiness of query processing.
- Hybrid approaches that combine computational methods with human knowledge with the aim to enhance, certify, and explain the outcomes of the main data-driven tasks, e.g., schema matching, and data curation and integration.
- Query processing able to interoperate during query execution. Furthermore, data quality assessment and bias detection methods are required in order to produce answers that ensure validity and trustworthiness.
- Methods capable of tracing data consumed from the selected sources, and explainable federated systems able to justify all the decisions made to produce the answer of a query over a federation of data sources.

The diversity of the problems that remain open presents enormous opportunities both in research and development. Advancement in this area will contribute not only more efficient tools but also solutions that users can trust and understand. As a result, we expect a paradigm shift in the area of big data integration and processing towards explainability and trustworthiness – issues that have thus far prevented global adoption of data-driven tools.