Received November 1, 2018, accepted November 26, 2018, date of current version January 29, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2888601

One Size Does Not Fit All: Querying Web Polystores

YASAR KHAN[1], ANTOINE ZIMMERMANN[2], ALOKKUMAR JHA[1], VIJAY GADEPALLY[3], MATHIEU D'AQUIN[1], AND RATNESH SAHAY[1]

[1]Insight Centre for Data Analytics, National University of Ireland Galway, H91AEX4 Galway, Ireland
[2]Univ Lyon, MINES Saint-Étienne, CNRS, Laboratoire Hubert Curien, UMR 5516, F-42023 Saint-Étienne, France
[3]Massachusetts Institute of Technology Lincoln Laboratory, Lexington, MA 02421-6426, USA

Corresponding author: Ratnesh Sahay (ratnesh.sahay@insight-centre.org)

This work was supported in part by the Science Foundation Ireland (SFI) under Grant SFI/12/RC/2289, in part by the European Regional Development Fund, and in part by the Agence Nationale de la Recherche under Grant ANR-14-CE24-0029.

ABSTRACT Data retrieval systems are facing a paradigm shift due to the proliferation of specialized data storage engines (SQL, NoSQL, Column Stores, MapReduce, Data Stream, and Graph) supported by varied data models (CSV, JSON, RDB, RDF, and XML). One immediate consequence of this paradigm shift results into data bottleneck over the web; which means, web applications are unable to retrieve data with the intensity at which data are being generated from different facilities. Especially in the genomics and healthcare verticals, data are growing from petascale to exascale, and biomedical stakeholders are expecting seamless retrieval of these data over the web. In this paper, we argue that the bottleneck over the web can be reduced by minimizing the costly data conversion process and delegating query performance and processing loads to the specialized data storage engines over their native data models. We propose a web-based query federation mechanism—called PolyWeb—that unifies query answering over multiple native data models (CSV, RDB, and RDF). We emphasize two main challenges of query federation over native data models: 1) devise a method to select prospective data sources—with different underlying data models—that can satisfy a given query and 2) query optimization, join, and execution over different data models. We demonstrate PolyWeb on a cancer genomics use case, where it is often the case that a description of biological and chemical entities (e.g., gene, disease, drug, and pathways) spans across multiple data models and respective storage engines. In order to assess the benefits and limitations of evaluating queries over native data models, we evaluate PolyWeb with the state-of-the-art query federation engines in terms of result completeness, source selection, and overall query execution time.

INDEX TERMS Databases, world wide web, query federation, query optimization, query planning, linked data, SPARQL, healthcare, and life sciences.

I. INTRODUCTION

The database experts have predicted the demise of "One Size Fits All" approach used in the data retrieval and management solutions [1]. This prediction is now evident, as in the last couple of years several data models (e.g., text, CSV, Graph, JSON, RDB, RDF, XML) and storage engines are proliferating with overlapping requirements, use-cases, and user-bases. This is particularly true in complex domains like healthcare and life sciences (HCLS) where the organisations are facing a major shift in the data retrieval requirements. A simultaneous use of different specialized data storage engines, data models, and supporting querying mechanism is needed to retrieve data from different interacting facilities (clinical measurement, medical history, laboratory test, demographics, etc.) [2]. In the presence of "too much data" and the paradigm shift in data storage and retrieval, it is now impractical to assume that the variety of high volume data residing in specialized storage engines will first be converted to a common data model, stored in a general-purpose data storage engine, and finally be queried over the Web.

On the same challenge of combining and querying data from several repositories, the central idea of Linked Data is to publish and link a wide variety of independent Web data silos in a manner that is queryable as one connected set of data sets supporting advanced Web uses, organisations, and scientific discoveries. The Linked Open Data (LOD) Cloud, as well as the enterprise applications of Linked Data, have shown success in connecting and querying resources across disparate data platforms [3]. There is wider availability of open-source and commercial tools that allow curating, publishing, aggregating, storing, and querying Linked Data. The typical linked data approach to query independent data silos is to convert all the underlying native data models into the RDF data model and devise querying mechanism through which these independent data silos can be queried in unison. While this approach can be practical for simpler verticals, in case of the HCLS domain it is already predicted that 2–40 exabytes of storage capacity will be needed by 2025 just for the human genomes which will continue to grow approximately 40 petabytes of additional genomic information each year [4]. Nevertheless, raw storage is not the main concern, but the amount of variant data (text, relational, stream, graph, etc.) being queried and analyzed is already seen as a major hurdle in the meaningful use of this vast amount of data.

The normal federation approaches evaluate a given query over multiple data stores complying to a single data model. Recently, an atypical approach of federating queries over heterogeneous data models has been initiated – called Polystore[1] [5] – that exploits different data models and storage engines in their native formats, i.e., without converting them to a common data model. The early demonstrators of Polystore have shown promising outcomes in federating queries across disparate data models used in the Multiparameter Intelligent Monitoring in Intensive Care II Databases (MIMIC II) [6]. It's important to note that, the concept of Polystore is an abstract idea of unifying querying over multiple data models which can be implemented using different technologies (RDB, Web, or Semantic technologies). In this paper, we present a semantic approach, called PolyWeb, to federate query over Web polystores containing cancer and biomedical data sets. We devise a query federation approach that focuses on source selection and joins over different data models (e.g., CSV, RDB, RDF). It is an open research problem to perform join across different data models. Further, it is important to understand the gain and loss of querying over data sources[2] in their native data models compared to the conventional approach of querying curated data sources – from several heterogeneous sources – using a common data model. We compare PolyWeb against the state-of-the-art SPARQL query federation engines.

The rest of the paper starts with the description of a motivation scenario that requires to query over disparate cancer genomics data sets. In the related work section, query federation approaches developed using relational database and semantic web methods are discussed. We then provide a brief overview of the PolyWeb architecture and discusses in detail

[1]http://wp.sigmod.org/?p=1629

[2]A data source encapsulates a data set complying to a particular data model/format (CSV, JSON, RDB, RDF, etc.)

```
PREFIX gene: <http://examp.ly/gene/>
PREFIX : <http://examp.ly/schema/>
SELECT * WHERE {
?cnv a :CNV;                                          t1
:chr ?chr;                                            t2
:start ?start;                                        t3
:end ?end;                                            t4
:sample ?sample;                                      t5
:gene gene:MYH7;                                      t6
:disease ?disease;                                    t7
:primary_site ?p_site;                                t8
:segment_mean ?seg_mean .                             t9
}
```

Listing. 1. Example SPARQL query.

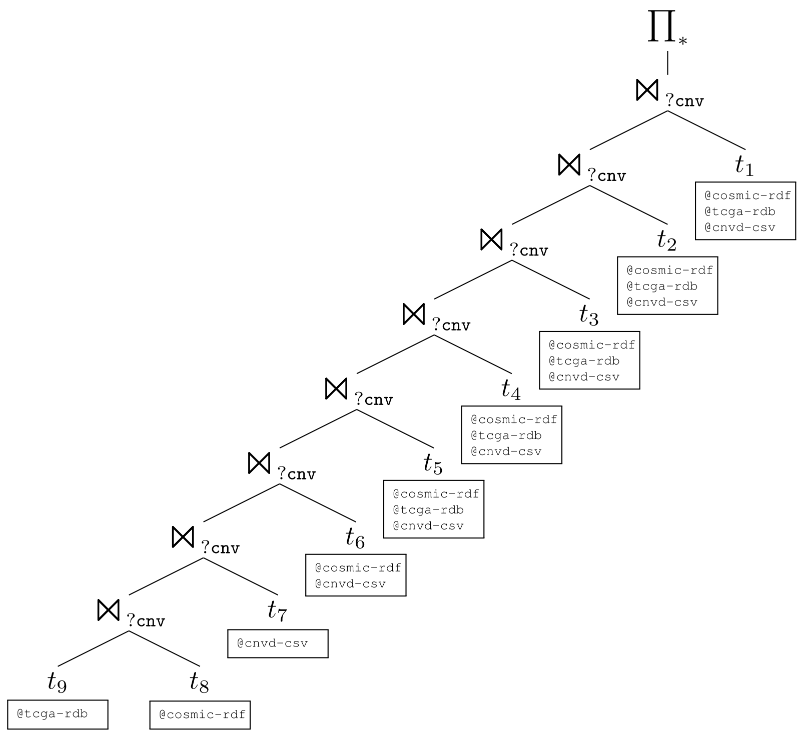
the two algorithms designed to select prospective data sets, optimize, plan and execute a given query. The correctness and complexity of the PolyWeb algorithms are also discussed. We provide an extensive evaluation of PolyWeb by comparing it with the state-of-art query federation engines. Finally, we conclude our work and provide future directions to further enhance our work.

II. MOTIVATING SCENARIO - CANCER GENOMICS

The Next Generation Sequencing (NGS) technologies are producing a massive amount of sequencing data sets. As said, there will be a top-up of approximately 40 petabytes of genomic information every year from a wide variety of data sources (hosting different databases, data formats, etc.) published by human genome research centers worldwide. Often, these data sets are published from isolated and different sequencing facilities. Consequently, the process of sharing and aggregating multi-site sequencing data sets are thwarted by issues, such as the need to discover relevant data from different sources, built scalable repositories, the automation of data linkage, the volume of the data and efficient querying mechanism. PolyWeb is motivated by the needs of the BIOOPENER project[3] which is aiming to link cancer and biomedical data sets providing interlinking and querying mechanisms to understand cancer progression from normal to diseased tissue with pathway components, which in turn helped to map mutations, associated phenotypes, and diseased pathways [7].

In cancer genomics, discoveries of biological and chemical entities (gene, pathway, drug, diseases, etc.) are available in several overlapping data sources containing complex genomic features, studies, and associations of such features. In order to understand the tumorigenesis, it is often the case that several genetic features, diseases, medical history, etc. are studied together. Considering the exponential growth and variety of genomics and biomedical data sets, it is impractical to assume that all these isolated and disparate data sets will be available in a single data model. In order to tap the vast knowledge locked in these data sets, it is now important to exploit them in native formats.

[3]http://bioopenerproject.insight-centre.org/



Π*
⋈ ?cnv
t1 @cosmic-rdf @tcga-rdb @cnvd-csv
t2 @cosmic-rdf @tcga-rdb @cnvd-csv
t3 @cosmic-rdf @tcga-rdb @cnvd-csv
t4 @cosmic-rdf @tcga-rdb @cnvd-csv
t5 @cosmic-rdf @tcga-rdb @cnvd-csv
t6 @cosmic-rdf @cnvd-csv
t7 @cnvd-csv
t8 @cosmic-rdf
t9 @tcga-rdb

FIGURE 1. **Unoptimised query plan involving three data sources and three data models (CSV, RDB, and RDF).**

The example SPARQL query shown in Listing 1 retrieves association of a gene (MYH7) with a primary site (where tumor progression starts), disease, copy number variation (CNV), CNV location (start, end), CNV segment mean, and reported samples of patients. The CNV information gives insight into the structural variation of a gene which helps analyses the progression of the cancer tumor, ultimately impacting on the prognosis and treatment of disease. Fig. 1 shows an unoptimised query plan that includes the type of databases or models (e.g., CSV, RDB, RDF) that can satisfy individual triples from three data sources. In our previous work, we developed a SPARQL federation engine – called SAFE [8] – that federates queries over genomics and clinical trial repositories represented in RDF. Similarly, in our previous work, we evaluated a wide variety of proposals (FedX, LHD, SPLENDID, FedSearch, GRANATUM, Avalanche, DAW, ANAPSID, ADERIS, DARQ, LDQPS, SIHJoin, WoDQA and Atlas) on how to execute SPARQL queries in federated settings [9]. However, we have no work to compare that federates over different native data models. Thus the motivation for our research is to investigate how to enable query federation over native data models in Web like open-world scenarios. As argued in the introduction, off-the-shelf federation engines cannot be used since they are designed to federate over a single data model. Hence following are the core research questions we address in this paper:

- How can we devise source selection in a Web like federated scenario where data sources comply with different data models?
- How can we implement a web-based query federation plan that retrieves correct and complete results from different data models?
- Can we optimize and execute the query federation plan for different native data models in a manner that allows us to compete with off-the-shelf RDF query federation engines?

The benefit of PolyWeb is two-fold: (i) It reduces the data conversion cost: a given query is executed over the native data models without converting multiple data models to a common data model. Therefore, the major advantage of PolyWeb holds in all the data retrieval scenarios where the data conversion cost is high. For instance, SPARQL-based federation engines listed in Table 1 execute a given query on the requirement (or assumption) that data sources are available in RDF; and (ii) it delegates data querying load to specialized data storage engines, instead of loading curated data from multiple data models to a general-purpose engine. In this article, we demonstrate that PolyWeb is helpful in reducing the data conversion cost and still be able to retrieve complete result sets from different native data models.

III. RELATED WORK

The scenario described in the previous section touches upon three main areas: source selection, query optimization and query execution in a federated environment. Both semantic web and relational database research communities have proposed and developed several federation systems that can unify query answering across disparate databases. The related work in these two broad areas is discussed in the following two sections.

TABLE 1. Overview of existing SPARQL query federation engines.

Systems	Source Selection	Join Type	Data Model	Code	Cache	Update
ADERIS [23]	index	nested loop	RDF	✗	✗	✗
ANAPSID [24]	query & index	adaptive	RDF	✗	✗	✓
Avalanche [25]	query & index	distributed, merge	RDF	✗	✓	✗
DARQ [26]	index	nested loop, bind	RDF	✗	✗	✗
DAW [27]	query & index	based on underlying system	RDF	✗	✓	✗
FedSearch [28]	query & index	bind, pull-based rank	RDF	✗	✓	✗
FedX [29]	query	nested loop, bind	RDF	✗	✓	✗
LHD [30]	query & index	hash, bind	RDF	✗	✗	✗
SPLENDID [31]	query & index	hash, bind	RDF	✗	✗	✗
HiBISCuS [32]	query & index	nested loop, bind	RDF	✗	✓	✓
FEDRA [33]	query & index	nested loop, bind	RDF	✗	✓	✓
SAFE [8]	query & index	nested loop, bind	RDF	✓	✓	✓
PolyWeb	query & index	nested loop, bind	CSV, RDB, RDF	✓	✗	✗

A. RELATIONAL DATABASES

The main focus of relational database approaches to query federation is around the closed-world enterprise settings which require a predefined number of data sources, schemas, and data sets to process queries across different business units within or between organisations. Some early [10] and more recent query federation approach [11] target the challenge of distributed query processing, distributed transactions, performance, and replica management. The database community realized the importance of managing distributed and heterogeneous databases and proposed ways to address the heterogeneity found across databases [12]. Several early data mediators [13] and middleware systems [14] – primarily based on the Global as View (GAV) and Local as View (LAV) approaches [15] – developed in the last three decades had the same motivation, namely, unifying the data retrieval and query process over different models and sources. The assumptions behind – such as availability of global schema, availability of mappings between schemas (local and global), and revising all the mappings with addition or removal of data sources – the GAV and/or LAV based approaches are too restrictive for Web like querying scenarios. Recently the database community is exploring a new perspective – called Polystore [5] – to unify queries over multiple data models. The early demonstrators of Polystore have shown promising outcomes in federating queries across disparate data models used in the Multiparameter Intelligent Monitoring in Intensive Care II Databases (MIMIC II) [6], [16], [17]. Similarly, NoSQL polyglot persistence [18] style solutions [19] are proposed to query over different data models. Domain specific solutions, such as streaming and sensor data processing [20], enterprise analytics [21], social media [22] have taken initiatives to query over multiple data stores which natively support different data models.

B. SEMANTIC TECHNOLOGIES

The semantic web community has proposed several query federation engines for a Web like scenario using SPARQL [9]. Table 1 shows a list of SPARQL-based federation engines and various querying features (e.g., source selection, join type, data model, code, cache, and update) supported by PolyWeb and other competing engines. Such engines accept an input query, decompose it into sub-queries, identify relevant data sources for sub-queries, send the sub-queries to the individual endpoints accordingly and at the end merge the final results for the query. The aim of such engines is to find and execute optimized query plans that minimize initial latency and total query execution times. This can be achieved by several factors; (i) using accurate source selection to minimize irrelevant messages, (ii) implementing efficient join algorithms, (iii) and using caching techniques to avoid repeated sub-queries. Source selection is typically enabled using a local index/catalogue and/or probing sources with queries at runtime. The former approach assumes some knowledge of the content of the underlying endpoints and requires update/synchronization strategies. However, the latter approach incurs a higher runtime cost, having to send endpoints queries to determine their relevance for various sub-queries. Thus, many engines support a hybrid approach of index and query-based source selection.

Initiatives, such as Ontology-Based Data Access (OBDA) exploit ontologies specifically for accessing relational databases (RDB) [34], [35] or semantic data lakes [36]; such initiatives are complementary to our PolyWeb proposal of querying different data models. The challenges of federating a given query over multiple data models are different in a Web like scenario. Often the data sets are selected from a large pool of prospective data sources and there is no control over the availability of complete schemas and data sets – unlike enterprises where data resources are controlled. It's important to note that database research is investigating the normal federation approach (using single RDB data model) *vs.* Polystore-based federation implemented using database technologies [5], [37]. Similarly, in this paper we are investigating normal federation (using single RDF data model) *vs.* Polystore-based federation implemented using semantic technologies. The obvious extension of our work is to investigate and compare database-style implemented Polystore *vs.* semantic technology implemented Polystore federation. This will help the wider research communities (database, semantic web, linked data) to understand the challenges of Polystore based query federation over different data models in a database-style enterprise like scenario vs. Web-like open scenario. To the best of our knowledge, no work has tackled the scenario of processing queries over a federation of different data models and sources; especially focusing on the source selection, optimization, planning, and execution of a query in a Web like scenario.

IV. POLYWEB

As discussed above, the query federation – either over single or multiple data models – is a well-studied problem over the last three decades. Different combination of methods (global schema, local schema, query translation, index-based, index-free, etc.) have been applied to unify query answering in a federated environment. In spite of several years of research, unfortunately in the current scenario, there is no off-the-shelf query federation engine available that functions over multiple data models in a Web environment. Our main innovation and contribution are to develop an open-source query federation engine that unifies query answering over multiple data models. Our approach is based on three key computational efforts: (i) creating mapping definitions of federated data sources; (ii) query translation between input and native queries query languages; and (iii) translation of query results. Our key aim is to reduce (or possibly avoid) the cost and effort of massive data conversion needed in a typical data warehousing and/or single data model approach for federating queries over different data sources. The PolyWeb architecture is summarized in Fig. 2, which shows its four main components: (i) Source Selection: performs source selection based on the capabilities of native data sources; (ii) Query Optimization: performs cost-effective arrangement of a query (triple patterns) in a manner that reduces query joins and remote requests on the network; (iii) Query Planning: builds a query plan based on joins found between query arguments; and (iv) Query Execution: performs the execution of sub-queries against the selected native data sources and merges the results returned. In addition to these components, PolyWeb can deal with the multiplicity of data models in the data sources by relying on mapping definitions that allow homogenising the interactions with databases in their native model. Then, the PolyWeb query federation method is presented into two algorithms, viz., Algorithm 2: PolyWeb Source Selection (SS), and Algorithm 3: PolyWeb Query Optimisation, Planning, and Execution (QOPE).

A. DATA MODELS AND MAPPING DEFINITIONS

The PolyWeb approach is dealing with multiple data models. In order to provide a generic account of how we can support many data models, we provide definitions that cover many

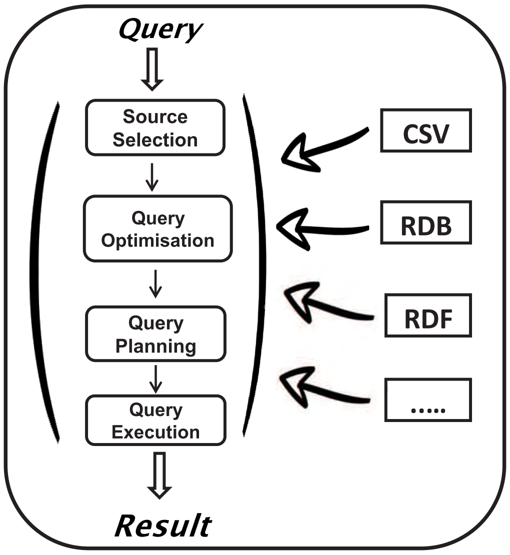


FIGURE 2. PolyWeb architecture.

cases. The important aspects are that a data model should provide a native query language and that it is possible to translate data sets in a data model to RDF, using a mapping definition. This section clarifies these notions, starting from a description of a data model.

Definition 1 (Data Model): A data model dm *defines a set* $\mathsf{DS}_{\mathsf{dm}}$ *of* data sets *that instantiate the data model, and a query language* $\mathsf{QL}_{\mathsf{dm}}$. *The query language allows one to pose a* query $q \in \mathsf{QL}_{\mathsf{dm}}$ *against a data set* $D \in \mathsf{DS}_{\mathsf{dm}}$ *to obtain a* response *in a* result format $\mathsf{RS}_{\mathsf{dm}}$. *A function* $\mathsf{Eval}_{\mathsf{dm}} : \mathsf{QL}_{\mathsf{dm}} \times \mathsf{DS}_{\mathsf{dm}} \to \mathsf{RS}_{\mathsf{dm}}$ *defines what response is obtained from issuing a query against a data set.*

An example of the data model is the relational model where data sets are relational databases (sets of relational tables), the query language corresponds to the relational algebra, given responses in the form of a relational table. We insist in particular on the RDF data model, where data sets are RDF graphs, that is, sets of RDF triples (s, p, o) where s is a blank node or an IRI, p is an IRI, and o is a blank node, an IRI, or a literal. The query language for RDF is SPARQL, which provides responses in the form of SPARQL results as defined by the SPARQL 1.1 standard [38]. We denote the RDF data model with rdf, the set of RDF graphs with $\mathcal{G}$, the set of SPARQL queries sparql, the set of SPARQL results as $\mathsf{RS}_{\mathsf{sparql}}$, and the evaluation of a SPARQL query q against an RDF graph G with $[\![q]\!]_G$.

In order to be able to deal with many data models in a seamless way using SPARQL-only queries, we need a way to express the translation between a data set in native data model into RDF. This is done with the notion of mapping definition that serves to define a transformation from a non-RDF data set to RDF.

Definition 2 (Mapping Definition): A mapping definition *for a data model* dm *is a function* $\mathsf{md} : \mathsf{DS}_{\mathsf{dm}} \to \mathcal{G}$.

A mapping definition can be used to query non-RDF data sets with the SPARQL query language. Indeed, if a SPARQL query q is posed on a data set $D \in \mathsf{DS}_{\mathsf{dm}}$ associated with a mapping definition md, then the response to the query can be defined as $[\![q]\!]_{\mathsf{md}(D)}$. However, this definition suggests that the data source should be completely translated to RDF according to the mapping definition, which would not be convenient when the data set is huge. Sometimes, it is not even possible to convert the source data because it is only made accessible via a query endpoint. Instead, the mapping definition should be used to define a query translation, defined as follows:

Definition 3 (Query Translation): A query translation *for a mapping definition* md *on a data model* dm *is a pair* $\langle \mathsf{qt}_{\mathsf{md}}, \mathsf{rt}_{\mathsf{md}} \rangle$ *where:*

- $\mathsf{qt}_{\mathsf{md}} : \mathsf{sparql} \to \mathsf{DS}_{\mathsf{dm}}$ *and*
- $\mathsf{rt}_{\mathsf{md}} : \mathsf{RS}_{\mathsf{dm}} \to \mathsf{RS}_{\mathsf{sparql}}$,

such that for all $q \in \mathsf{sparql}$ *and all* $D \in \mathsf{DS}_{\mathsf{dm}}$, $[\![q]\!]_{\mathsf{md}(D)} = \mathsf{rt}_{\mathsf{md}}(\mathsf{Eval}_{\mathsf{dm}}(\mathsf{qt}_{\mathsf{md}}(q), D))$.

Mapping definitions can be written in dedicated mapping languages, such as R2RML [39] (for mapping relational databases to RDF), RML [40] (extending R2RML to other data models, such as XML, JSON, CSV, HTML), XSPARQL [41] (initially for XML), SPARQL-Generate [42] (CSV, JSON, XML, HTML, CBOR), XSLT [43] (if limited to transformations that results in RDF/XML), defined as a JSON-LD [44] `@context` for JSON, or using the CSVW vocabulary [45] for CSV.

B. DATA SUMMARIES

PolyWeb's data-summary is a light-weight index that stores minimal information about data sources. PolyWeb's data-summary generation algorithm is shown in Algorithm 1. The algorithm takes the set of all data sets as input and generates a concise data summary that enables source selection, query optimization, planning, and evaluation. By proposing data-summary algorithm, we claim that PolyWeb has significantly improved data-summary generation times when compared to other index-based approaches, for instance, HiBISCuS [32]; this allows for faster re-computation of summaries when the underlying sources change (Table 7). The data summaries and their generation algorithm are described in the following.

We assume a set of data sets $\mathcal{D}$ where each data set $D \in \mathcal{D}$ belongs to different data models (e.g., rdf, rdb, csv). We denote by $\mathrm{preds}(D) := \{p \mid \exists s, o : (s, p, o) \in D\}$ the set of all predicates in D. We denote $\mathrm{concepts}(D) := \{o \mid \exists s, p : p = \mathit{rdf{:}type} \wedge (s, p, o) \in D\}$ as the set of all classes referenced in D. We assume that each data set is published as an endpoint at a specific location, where $\mathrm{loc}(D)$ denotes location (endpoint URL) of the data set D.

For each data set $D \in \mathcal{D}$, PolyWeb stores the following information in data summaries; (i) the endpoint or data access point URL, denoted by $\mathrm{loc}(D)$, which represents the location of data set D (*line 7* of Algorithm 1); (ii) a set of distinct predicates $\mathrm{dpreds}(D)$ in data set D (*lines 9 and 16* of Algorithm 1); and finally (iii) a set of unsafe predicates $\mathrm{upreds}(D) \subseteq \mathrm{dpreds}(D)$ in each data set D (*lines 14 and 19* of Algorithm 1). PolyWeb's data-summaries for each data set D is computed in two steps: (i) first we compute a set of distinct predicates $\mathrm{dpreds}(D) \subseteq \mathrm{preds}(D)$ for each data set D. For RDF data sets, we extract a set of distinct predicates IRIs found in a given data set D: $\mathrm{dpreds}(D)$ by directly querying the data set and store it locally in data summaries. In the case of non-RDF data sets, a set of mapping definitions $\mathsf{md}(D_1, \ldots, D_n)$ (discussed in the Section IV-C) are used to associate predicate IRIs to non-RDF data sets. The mapping definitions provide RDF view of the non-RDF data models that are stored in data summaries; and (ii) second, we compute the set of unsafe predicates $\mathrm{upreds}(D) \subseteq \mathrm{dpreds}(D)$ from each data set.

PolyWeb proposes a notion of "unsafe predicate" to optimize query execution when two or more triple patterns of a BGP[4] are grouped together for evaluation against federated data sets (discussed in the Section IV-C). A group of triple patterns in a BGP are evaluated as a conjunctive query where all triple patterns must match together against evaluating data sets. However, in practical scenarios, a regular complete structures of triples cannot be assumed in all data sets. It is important in a setting of federated data sets, that a given query (BGP) retrieves results where triples in a data set are available, but do not reject (or eliminate) the solution because some part (or portion) of the query pattern does not match. Thanks to the SPARQL OPTIONAL construct, which does not eliminates the solution when the optional part does not match. But, it is not a straightforward task to know in advance which triple pattern(s) of a BGP might have an unmatched solution (triples) in a data set. Also, the unlimited use of optional parts in a SPARQL query could lead to high complexity [46]. The purpose of computing a subset of unsafe predicates $\mathrm{upreds}(D) \subseteq \mathrm{dpreds}(D)$ is to filter out all those predicates – before the actual query execution – that may have an unmatched solution (triples) in a data set. An unsafe predicate may cause result incompleteness when a triple pattern (holding the unsafe predicate) is grouped together with one or more triples patterns of a BGP.

For instance, the Listing 2 shows two example data sources (D1, D2) and their corresponding data summaries computed by the Algorithm 1 are shown in the Listing 3. The first data source (D1) describes entities (`:cnv-1-E1`, `:cnv-2-E1`, etc.) of type Copy Number Variation (CNV). The CNV information gives insight into the structural variation of a gene. The description of CNV type entities has a regular triple structure which uses following predicates: `[:type, :gene, :start, :end, :sample, :chr]`. However, the Triple 1.25 in the Listing 2 with object `:chr-3` has no label description, which means, grouping together Pattern 1 and Pattern 2 of the example BGP-1 (in the Listing 4) will result in the elimination of solution Triple 1.25 in the data source D1.

We mark predicate `:label` in the data source D1 as unsafe predicate because there is at least one subject (entity)

[4] A Basic Graph Pattern (BGP) is a set of Triple Patterns

Algorithm 1 PolyWeb Data Summaries Generation

```
1  $\mathcal{D} = \{D_1, \ldots, D_n\}$ ;                                   /* Set of all data sources */
2  loc(·) ;                                                            /* Mapping to endpoint URLs */
3  $\mathsf{md}(D_1, \ldots, D_n)$ ;                             /* Mapping definitions for non-RDF data sets */
4  $S \leftarrow \{\}$ ;                                       /* PolyWeb summaries to be retrieved as output */
5  for each $D \in \mathcal{D}$ do                                      /* for each data source in $\mathcal{D}$ */
6      initialise $S_D$ ;                                                 /* initialise data summary */
7      $S_D$.setURL(loc($D$)) ;                                           /* map data endpoint URLs */
8      if $D \in \mathsf{DS}_{rdf}$ then                                       /* if data set is RDF */
9          $S_D$.dpreds($D$) $\leftarrow \{p \in \mathrm{preds}(D) \mid \nexists p' : p' = p \wedge p' \in \mathrm{preds}(D)\}$ ;  /* find distinct predicates in D */
10         $S_D$.dconcepts($D$) $\leftarrow \{o \in \mathrm{concepts}(D) \mid \nexists o' : o' = o \wedge o' \in \mathrm{concepts}(D)\}$ ;  /* find distinct classes in D */
11         for each distinct classes $C \in$ dconcepts($D$) do
12             $dpred_c[C] \leftarrow \{p \in \mathrm{dpreds}(C) \mid domain(p) \in C\}$ ; /* find distinct predicates per distinct class */
13             if any subject $s \in C$ exists without a distinct predicate $p \in dpred[C]$ then     /*  */
14                 $S_D$.add_upreds($p$) ;                                /* add unsafe predicate */
15     else                                                               /* if data set is non-RDF */
16         $S_D$.dpreds($D$) $\leftarrow \{p \in \mathrm{preds}(\mathsf{md}(D)) \mid \nexists p' : p' = p \wedge p' \in \mathrm{preds}(\mathsf{md}(D))\}$
           /* scan and add distinct predicates for $D$ from mappings definition $\mathsf{md}(D)$ */
17         for each distinct predicates $p \in$ dpreds($D$) do
18             if corresponding column (p) in table has a NULL value then          /*  */
19                 $S_D$.add_upreds($p$) ;                                /* add unsafe predicate */
20     $S \leftarrow S \cup \{S_D\}$
21     return $S$ ;                                                   /* Data summaries of all sources */
```

e.g., `:chr-3` of the Triple 1.25, for which the predicate `:label` is missing and that leads to result incompleteness for queries like the BGP-1 (Listing 4). We identify such unsafe predicates early, as part of the data summary, which is further used during the query optimization process (discussed in the Section IV-D). It is important to note that the object `:chr-3` of Triple 1.25 is further described in the data source D2 in Triple 2.20. Therefore, in a federated setting, it is crucial not to eliminate solution because a matching solution may exist in other remote data sets. So, both the triple patterns of the example BGP-1 (Listing 4) should be evaluated independently without grouping them together to ensure result completeness. Similarly, `:cnv-4-E1` has no sample description in data source D1 but can be found in data source D2 (Triple 2.26). Therefore grouping together Pattern 1 and Pattern 2 of the example BGP-2 will result in the unevaluated Triple 1.32 of the data source D1. In this case, `:sample` is the unsafe predicate. PolyWeb exploits the unsafe predicates later at the query optimization process to decide which triple patterns are safe to be grouped together ensuring completeness of a query result.

The unsafe predicates are computed in a four step process by the data summary Algorithm 1: (i) first the algorithm retrieves the type information (e.g., `:cnv-1-E1 a :CNV`) available in a data set. The data summary algorithm extracts all the distinct types (i.e., classes) that exist in a data set. For example, the data source D1 has three types: `:CNV`, `:Chromosome` and `:Gene`; (ii) second, the algorithm extracts a set of distinct predicates that describes entity of a particular class (type). For example, the `:CNV` type in data source $D1$ has following distinct predicates: `[:type, :gene, :start, :end, :sample, :chr]`; (iii) third, the algorithm scans to identify if there exist any entity (subject), where any of its predicates identified in the Step 2, is missing. If there exists a missing predicate for an entity of a particular type, then it is marked as unsafe. For example, the predicates (`:label` and `:sample`) are unsafe because they are missing for entities `:chr-3` (Triple 1.25) and `:cnv-4-E1` (Triple 1.27) of the data source $D1$. The predicates (`:type`, `:gene`, `:start`, `:end`, and `:chr`) that describe the entities of type `:CNV` are marked as safe because every entity uses them. The same process is repeated for other types `: Chromosome` and `:Gene` in the data source D1. The set of unsafe predicates is empty for the data source D2, as there exists no such predicate, which is missing for any entity of its corresponding type; finally (iv) for entities whose type information is missing their corresponding predicates are considered unsafe as we cannot

```
# First data source (D1)
:cnv-1-E1 rdf:type :CNV.                        Triple 1.1
:cnv-1-E1 :gene :LCE3C.                         Triple 1.2
:LCE3C rdf:type :Gene.                          Triple 1.3
:cnv-1-E1 :start 152583052.                     Triple 1.4
:cnv-1-E1 :end 152613763.                       Triple 1.5
:cnv-1-E1 :sample "sample-1".                   Triple 1.6
:cnv-1-E1 :chr :chr-1.                          Triple 1.7
:chr-1 rdf:type :Chromosome.                    Triple 1.8
:chr-1 :label "1".                              Triple 1.9
:cnv-2-E1 rdf:type :CNV.                        Triple 1.10
:cnv-2-E1 :gene :IFT172.                        Triple 1.11
:IFT172 rdf:type :Gene.                         Triple 1.12
:cnv-2-E1 :start 41527.                         Triple 1.13
:cnv-2-E1 :end 69904437.                        Triple 1.14
:cnv-2-E1 :sample "sample-2".                   Triple 1.15
:cnv-2-E1 :chr :chr-2.                          Triple 1.16
:chr-2 rdf:type :Chromosome.                    Triple 1.17
:chr-2 :label "2".                              Triple 1.18
:cnv-3-E1 rdf:type :CNV.                        Triple 1.19
:cnv-3-E1 :gene :CTDSPL.                        Triple 1.20
:CTDSPL rdf:type :Gene.                         Triple 1.21
:cnv-3-E1 :start 37940617.                      Triple 1.22
:cnv-3-E1 :end 37945438.                        Triple 1.23
:cnv-3-E1 :sample "sample-3".                   Triple 1.24
:cnv-3-E1 :chr :chr-3.                          Triple 1.25
:chr-3 rdf:type :Chromosome.                    Triple 1.26
:cnv-4-E1 rdf:type :CNV.                        Triple 1.27
:cnv-4-E1 :gene :CDKN2A.                        Triple 1.28
:CDKN2A rdf:type :Gene.                         Triple 1.29
:cnv-4-E1 :start 21918582.                      Triple 1.30
:cnv-4-E1 :end 22388823.                        Triple 1.31
:cnv-4-E1 :chr :chr-9.                          Triple 1.32
:chr-9 rdf:type :Chromosome.                    Triple 1.33
:chr-9 :label "9".                              Triple 1.34

# Second data source (D2)
:cnv-1-E2 rdf:type :CNV.                        Triple 2.1
:cnv-1-E2 :gene :UGT2B17.                       Triple 2.2
:UGT2B17 rdf:type :Gene.                        Triple 2.3
:cnv-1-E2 :start 68494771.                      Triple 2.4
:cnv-1-E2 :end 68639398.                        Triple 2.5
:cnv-1-E2 :sample "sample-4".                   Triple 2.6
:cnv-1-E2 :chr :chr-4.                          Triple 2.7
:chr-4 rdf:type :Chromosome.                    Triple 2.8
:chr-4 :label "4".                              Triple 2.9
:cnv-2-E2 rdf:type :CNV.                        Triple 2.10
:cnv-2-E2 :gene :FBXL7.                         Triple 2.11
:FBXL7 rdf:type :Gene.                          Triple 2.12
:cnv-2-E2 :start 15719113.                      Triple 2.13
:cnv-2-E2 :end 15720369.                        Triple 2.14
:cnv-2-E2 :sample "sample-5".                   Triple 2.15
:cnv-2-E2 :chr :chr-5.                          Triple 2.16
:chr-5 rdf:type :Chromosome.                    Triple 2.17
:chr-5 :label "5".                              Triple 2.18
:chr-3 rdf:type :Chromosome.                    Triple 2.19
:chr-3 :label "3".                              Triple 2.20
:cnv-4-E1 rdf:type :CNV.                        Triple 2.21
:cnv-4-E1 :gene :CDKN2A.                        Triple 2.22
:CDKN2A rdf:type :Gene.                         Triple 2.23
:cnv-4-E1 :start 21918582.                      Triple 2.24
:cnv-4-E1 :end 22388823.                        Triple 2.25
:cnv-4-E1 :sample "sample-6".                   Triple 2.26
:cnv-4-E1 :chr :chr-9.                          Triple 2.27
:chr-9 rdf:type :Chromosome.                    Triple 2.28
:chr-9 :label "9".                              Triple 2.29
```

Listing. 2. Example data sources.

TABLE 2. Example of tabular data (CNV Table).

id	gene	start	end	sample	chr
cnv-1-E1	LCE3C	152583052	152613763	sample-1	1
cnv-2-E1	IFT172	41527	69904437	sample-2	2
cnv-3-E1	CTDSPL	37940617	37945438	sample-3	3
cnv-4-E1	CDKN2A	21918582	22388823	NULL	9

```
# Data summaries of D-1
Distinct Predicates = [ :type, :gene, :start, :
    end, :chr, :sample, :label ]
Unsafe Predicates = [ :sample, :label ]

# Data summaries of D-2
Distinct Predicates = [ :type, :gene, :start, :
    end, :chr, :sample, :label ]
Unsafe Predicates = [ ]
```

Listing. 3. Data summaries Example.

```
# Basic Graph Pattern-1 (BGP-1)
?cnv :chr ?chr. # [D1, D2] Pattern 1
?chr :label ?label. # [D1, D2] Pattern 2

# Basic Graph Pattern-2 (BGP-2)
?cnv :chr ?chr. # [D1, D2] Pattern 1
?cnv :sample ?sample. # [D1, D2] Pattern 2
```

Listing. 4. A federated query example.

execute the above four steps if type (class) information is unavailable. It is not uncommon in the real-life RDF data sets that schema coreference links (e.g., `:cnv-1-E1 a :CNV`) are missing and therefore, PolyWeb has a limitation that it can only identify safe (or unsafe) predicates when the type information is available in a data set.

In case of the non-RDF data sets, unsafe predicates are identified by using the mapping definitions. The Table 2 shows RDF triples (data source $D1$) of the Listing 2 in the tabular format. In case of the tabular data set (RDB or CSV), the predicate and type information are mapped to the corresponding source (table) and column. The table or source information is available from the `rr:logicalTable` (R2RML mapping in the Listing 5) or `rml:logicalSource` values, e.g., `genome:start`, source is `cnvd` (RML mapping in Listing 6). In order to find the unsafe predicates, below three steps are executed: (i) first, distinct types (classes) and their corresponding predicates are extracted from mapping definitions. For example, `genome:CNV` found in the `rr:subjectMap` of triple map `#TriplesMap1` (Listing 5) and `#TriplesMap2` (Listing 6), is a distinct type; (ii) second, the corresponding predicates (`genome:start`) for this type (`genome:CNV`) is extracted from the `rr:objectMap` found in `rr:predicateObjectMap`; (iii) third, the non-RDF data sets are queried to check, if there exists a single record (row) for which any column holds a `NULL` value. If any column with a `NULL` value is found, then that particular column (or mapped predicate) is marked as unsafe. For instance, the Table 2 shows that the `sample` column has a `NULL` value in the last row, therefore, it is marked unsafe. Consequently, a triple pattern of a

```
<#TriplesMap1> a rr:TriplesMap;
rr:logicalTable [ rr:tableName "cnv" ];
rr:subjectMap [
rr:template "http://insight.org/./{id}";
rr:class genome:CNV
];
rr:predicateObjectMap [
rr:predicate genome:chr ;
rr:objectMap [ rr:column "chr" ];
]
].
```

Listing. 5. R2RML mapping.

```
<#TriplesMap2> a rr:TriplesMap;
rml:logicalSource [ rml:source "cnvd" ];
rr:subjectMap [
rr:template "http://insight.org/./{id}";
rr:class genome:CNV
];
rr:predicateObjectMap [
rr:predicate genome:start;
rr:objectMap [ rml:reference "start" ]
]
].
```

Listing. 6. RML mapping.

BGP containing the `sample` predicate will not be grouped together with any other triple pattern for evaluation (see BGP-2 of the Listing 4).

The indexes (loc(D), upreds(D), and dpreds(D)) that we compute in data summary will be used in the following source selection, query optimisation, planning, and execution algorithms.

C. SOURCE SELECTION

PolyWeb performs a predicate-based source selection where a set of relevant data sources for a given query are discovered on the match between predicates used in a basic graph pattern[5] (*bgp*) and input data sources. A single basic graph pattern is defined as a sequence of triple patterns, with optional filters. PolyWeb performs the sources selection on top of different data models and sources, e.g., SPARQL end-points, RDB and CSV data-access points. PolyWeb relies on mapping definitions, such as R2RML [39] and RML [40] mappings to identify relevant data sources. In our work, a mapping definition provides an RDF view of non-RDF data such that non-RDF data can be queried with SPARQL. Listings 5 and 6 show example snippets of R2RML and RML mappings of our motivational scenario. For instance, the R2RML mapping shows that an RDF predicate *genome* : *chr* is mapped to a database column *Chromosome* in table *cnv* and the RML definition shows a mapping from RDF predicate *genome* : *start* to a CSV column *Start_Position* of the *cnvd* table.

R2RML (RDB to RDF Mapping Language) [39] is a W3C standard for expressing customized mappings from relational databases to RDF. RML (RDF Mapping Language) [40] extends R2RML applicability to expressing mappings from CSV, HTML, JSON, and XML to RDF. Mappings defined using R2RML and RML are itself RDF graphs. PolyWeb employs a simple data-free[6] indexing mechanism where the index stores minimal information about given data sets. In the case of RDF data sets, it stores only predicate IRIs obtained from individual data set; otherwise, it exploits the mapping definitions to associate predicate IRIs to non-RDF data sets. We formalise this and present formal notations in the following definitions:

Definition 4 (Source Selection): In a federated query environment, given a triple pattern t of a bgp in a query Q executed against data sets $\mathcal{D}$, the set of relevant sources for t in $\mathcal{D}$ is the set $\mathcal{R}_t \subseteq \mathcal{D}$ of data sets that can provide answers when queried with t. We use the notation $\mathcal{R}_T$ to denote the family $(\mathcal{R}_t)_{t \in T}$ for a set of triple patterns T, and use $\mathcal{R}$ when the context does not require specifying the triple patterns.

PolyWeb uses an index of the predicates and an index of the mapping definitions for all data sets. Thanks to the mapping definitions, it is possible to identify non-RDF data sources that can return results from a SPARQL query. The input SPARQL query is broken down into BGPs. Each BGP is decomposed into triple patterns from which we get either a bound predicate p (i.e., an IRI), or an unbound predicate (i.e., a variable). Algorithm 2 shows the source selection for a *bgp*. If the predicate is bound (tested at Line 5), then we make use of the index to select relevant sources, by querying the predicates sets of each data set; if the predicate is a variable (Line 8), we issue an ASK query to verify if the data set is relevant (Line 11).

The current version of PolyWeb supports only single BGPs (excluding constructs, such as OPTIONAL, UNION, etc.), and does not issue an ASK query in case all parts of the triple are variables. It also only supports the mapping languages R2RML and RML. We now discuss in detail the operation of the algorithm. The source selection algorithm will return a set of relevant sources for each triple pattern; these will be stored in $\mathcal{R}_t$. The sources relevant for each t will be kept separate in the results since different sources may be selected for the same triple pattern. In the case of Example 1, which contains one *bgp* with nine triple patterns. This loop will iterate once, since we have a single *bgp* in the example query. *Lines 3–5*: the algorithm proceeds by iterating over each triple pattern t contained in the set of triple patterns $\mathcal{T}$ for the current *bgp*. The algorithm then extracts predicates p from each triple pattern and checks whether the predicate is bound or not. *Line 5–Line 7*: if the predicate p is bound, then the algorithm queries whether the predicate is present in the predicates sets of each data set. In this way, it checks the presence of each predicate in all the available data sets $\mathcal{D}$.

In our motivating example, the predicates in the first five triple patterns are covered by all the three data sets, hence these three data sets are added to the relevance set ($\mathcal{R}_t$). The predicate *gene* in the sixth triple pattern is covered by two data

[5]https://www.w3.org/TR/sparql11-query/#BasicGraphPatterns

[6]index stores only predicate IRIs

Algorithm 2 PolyWeb Source Selection (SS)

```
1  $\mathcal{D}$, md$(D_1, \ldots, D_n)$, $bgp$ ;          /* data sets, mapping definitions, basic graph pattern */
2  $\mathcal{R}_t \leftarrow \{\}$ ;                                        /* set of relevant data sources */
3  for each $t \in bgp$ do                                    /* for each triple pattern in $bgp$ */
4      $p \leftarrow p \in t \in bgp$ ;                                      /* get predicate from $t$ */
5      if $p$ is bound then                                        /* if predicate is bound */
6          if $p \in$ preds(md$(D_1, \ldots, D_n)$) then               /* if data sets cover predicate */
7              $\mathcal{R}_t \leftarrow \mathcal{R}_t \cup \{D\} \cup \{$md$(D)\}$ ;   /* store the selected source and mapping definition
               for $t$ */
8      else                                                    /* if predicate is unbound */
9          for each $(s, p, o) \in bgp$ such that $p$ is unbound $\wedge$ ($s$ is bound $\vee$ $o$ is bound) do
10             for each $D \in \mathcal{D}$ do                                  /* for each data set */
11                 if $ASK((s, p, o), D) = true$ then          /* use ASK query to check relevance */
12                     $\mathcal{R}_t \leftarrow \mathcal{R}_t \cup (\{(s, p, o)\} \cup$ md$(D))$ ;    /* add all graphs and mapping definitions
                       for $D$ */
```

sets (CNVD and COSMIC). The predicates in last three triple patterns (i.e., *disease*, *primary_site* and *segment_mean*) are covered uniquely by three data sets (CNVD, COSMIC and TCGA). Therefore, the relevance set for the last three triple patterns contains one data set per triple pattern (e.g., *disease* contained in CNVD, *primary_site* contained in COSMIC, and *segment_mean* contained in TCGA).

In order to increase the selectivity of source selection, for a triple pattern where a predicate is unbound and subject or object is bound, the algorithm sends an ASK query to data set of RDF type to see if it may be relevant or not (*Line 11*). In our implementation, we only issue a ASK query when either the subject or object are bound, which is sufficient if we assume that the data sets are not empty. For the experiments, we rely on an implementation that do not yet allow the ASK construct to be applied to non-RDF data sources. In the future version, we plan to devise a method using "SQL EXISTS" – similar to the SPARQL ASK semantics – that can probe non-RDF data sets when predicates are unbound.

D. POLYWEB QUERY OPTIMISATION, PLANNING AND EVALUATION

PolyWeb (Algorithm 3) creates a federated query plan against the relevant data sources and executes that plan to get results from multiple data models and merge it into a single result set. The algorithm takes relevance set $\mathcal{R}_t$ obtained from the source selection algorithm 2 and mappings $\mathcal{M}$ (only RML and R2RML mappings in our implementation at the time of writing), as an input and returns a merged result set $\mathcal{RS}$ for the input SPARQL query.

1) QUERY OPTIMISATION

We propose a predicate-based join group (PJG) method to optimize the execution of federated queries. PJG reduces the cost of federated query processing by minimizing the number of local joins, a key factor which influences the evaluation of a given query. The more triple patterns are added to PJG and sent as a single query, the more local joins are minimized. FedX [29] proposed the idea of exclusive groups to combine (or group) triple patterns against a relevant data source that can satisfy them, which help minimizing the local joins and the execution cost of the input query. FedX approach is index-free which exploits the SPARQL UNION construct to aggregate the partial results obtained for each triple patterns in an exclusive group. This index-free approach of grouping the triple patterns requires a high amount of remote requests and data transferred on the network. Similarly, HiBISCuS [32] depends on FedX for the actual execution of queries, therefore, a grouping of triple patterns identified in the input SPARQL query remains the same as FedX.

Definition 5 (Predicate-Based Join Group): Let $T = \{t_1, \ldots, t_n\}$ *a set of triple patterns of a BGP and for each* $t_i \in T$, $\mathcal{R}_{t_i}$ *be the set of relevant data sets for triple pattern* t_i, *as computed by Algorithm 2 from* $\mathcal{D}$ *and* md. *We define the predicate-based join group (PJG)* $\Sigma_{R_T} = \{T \mid p \in \{\text{preds}(T) \setminus \text{upreds}(R_T)\}\}$ *as the group of triple patterns* T *that can be evaluated against a group of relevant data sets* R_T. *PJG works on a condition that* $\text{preds}(T) \cap \text{upreds}(R_T) = \emptyset$, *which means that grouping of triples* T *is only possible when none of the predicates in* preds(T) *overlap with the unsafe predicates of relevant data sets* upreds(R_T).

In case of FedX [29] an exclusive group can only be exploited when it contains triple patterns of size ≥ 2, hence, for our motivating example FedX creates nine (9) join groups with eight (8) joins, each join group contains only one triple pattern (see Fig. 1). The proposed predicate-based join group (PJG) identifies a group of triple patterns against a group of relevant data sets (unlike FedX that identifies a group of triple patterns against a single data set), which generates 6 predicate-based join groups (see Table 3) of triple patterns and 5 joins (see Fig. 3) after executing the PolyWeb's query optimisation, planning and execution (QOPE) algorithm 3. Hence, PolyWeb reduces the number of local joins evaluated

TABLE 3. PolyWeb: Example of predicate-based join group (Σ_6).

Predicate-based Join Group (PJG)	Triple Patterns	Data Set (Data Model)
Σ_1	t_6	COSMIC (RDF), CNVD (CSV)
Σ_2	t_7	CNVD (CSV)
Σ_3	t_8	COSMIC (RDF)
Σ_4	t_9	TCGA (RDB)
Σ_5	t_5	COSMIC (RDF), TCGA (RDB), CNVD (CSV)
Σ_6	t_1, t_2, t_3, t_4	COSMIC (RDF), TCGA (RDB), CNVD (CSV)

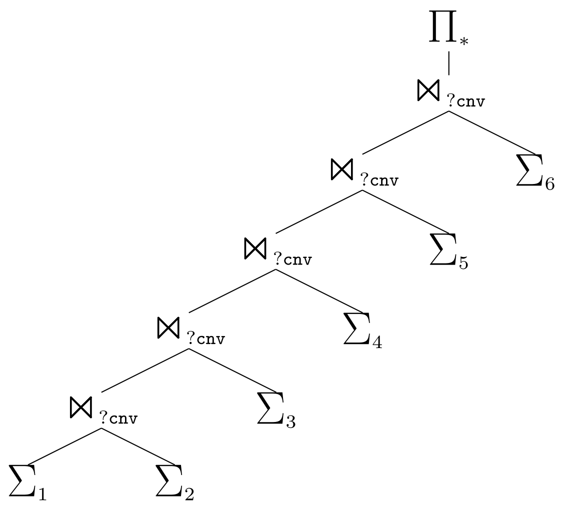


FIGURE 3. PolyWeb: Optimized federated query plan of example query.

at the federation engine level. The group of triple patterns identified in a PJG are evaluated at relevant endpoints and the results obtained from each relevant endpoint are merged as a union. For instance, Σ_6 is sent as single query to each relevant endpoint (i.e., COSMIC, TCGA and CNVD) and the results returned from each endpoint are merged as a union. PJG is a light-weight index-based (i.e., data summary) approach to group triple patterns of a BGP in manner that competes with normal (i.e., single data model) query federation engines and still retrieves correct and complete results from multiple native data models. PJG is created to achieve result completeness when two or more triple patterns are grouped together for evaluation over relevant data sets R. We now discuss in detail the operation of PolyWeb QOPE algorithm. For our running example query all the three data sets (namely, TCGA, COSMIC and CNVD) are identified as relevant by the SS algorithm 2.

Algorithm 3 creates predicate-based join groups $(\sum_i)_{i=1}^n$ for triple patterns of a BGP along with their execution cost in an ascending order. The function *PJG* (Line 5) takes a BGP, relevant sources, distinct predicates, and unsafe predicates as parameters and generate an array of predicate-based join groups $((\sum_i)_{i=1}^n)$ for each distinct set of data sets, and keeps the data sets associated with them. Costs are stored in a separate array (Line 7) used for ordering the predicate-based join groups (Line 8). Fundamentally, there are two ways cost be calculated [47] for the query optimisation (i) static: cost calculated based on specific patterns in a BGP before actually executing a query; and (ii) dynamic: certain portions of a query (i.e., sub-query) are executed against the data sources to calculate a more accurate cost of executing the query. We opted for the first approach because it is less expensive computationally, as the second approach would require query and result transformations. We base our cost calculation on the variable counting approach [47]. The cost of a predicate-based join group is the sum of the costs of all triple patterns in that group (*Line 7*). The more variables there are in a triple pattern, the higher the cost is.

Lines 9–15: Once the predicate-based join groups $((\sum_i)_{i=1}^n)$ are constructed, the algorithm iterates over them and create a federated query plan. The federated query plan is stored in $\overrightarrow{QP}$, which is initialised on *Line 9*. Then the algorithm starts finding join variables between the predicate-based join groups.

2) QUERY PLANNING AND EXECUTION

The join variables between two predicate-based join groups, are denoted by $\bowtie_{var}$ and is calculated at *Line 12* by identifying the common variables between them. For instance, predicate-based join groups Σ_1 and Σ_2 have a common variable i.e., *?cnv* therefore join is possible between these two groups. The federated join is created by assigning the left and right arguments (*Line 14*) of a query plan. For our running example, Σ_1 is assigned as a left argument and Σ_2 as a right argument of the join i.e., $\Sigma_1 \bowtie \Sigma_2$. Similarly, the algorithm checks for joins between the remaining predicate-based join groups and the query plan is updated accordingly. For our running example, all the five joins exist on the variable *?cnv* and the prepared query plan is shown in Fig. 3. Once a query plan ($\overrightarrow{QP}$) is constructed, it starts executing in a bottom-up fashion, i.e., it starts from the leaf nodes (Line 16) and traverses up the tree until a root node is reached to generate a combined result set $\mathcal{RS}$ for the input query. The joins are physically implemented in a nested loop bind fashion. Since two predicate-based join groups do not contain any common triple pattern, a simple implementation of nested loop bind is achieved by first materializing the left argument and binding the results of left argument with the right argument based on a set of identified join variables. As PolyWeb federates over different data models, the query execution component uses different transformations between queries and result sets. In our implementation (*Translate.Query*(.) in Line 19), we have the following translations (i) **SPARQL to SQL**: If an argument of the query plan needs to execute over RDB data set, then the argument (sub-query) is translated in the SQL format. For instance, an example SPARQL query and its translation to SQL and Apache Drill SQL queries as well as individual result sets are

Algorithm 3 PolyWeb Query Optimisation, Planning and Execution (QOPE)

```
1  $bgp, \mathcal{R}_t$, md ;    /* basic graph pattern, relevant data sources, mapping definitions */
2  dpreds($\mathcal{D}$), upreds($\mathcal{D}$) ; /* distinct predicates in all data sets, unsafe predicates in all
   data sets */
3  $\mathcal{RS} \leftarrow \{\}$ ;                          /* initialise query result set */
4  $\Sigma_{\mathcal{R}_T} \leftarrow \{\}$ ;                /* initialise predicate-based join group (PJG) */
   /* Step 1: Query Optimisation                                                             */
5  $(\sum_i)_{i=1}^n \leftarrow PJG(bgp, \mathcal{R}_t, \text{dpreds}(\mathcal{D}), \text{upreds}(\mathcal{D}))$ ; /* build predicate-based join groups (PJGs) */
6  for each $\sum_i \in (\sum_i)_{i=1}^n$ do                 /* for each PJG */
7      $\text{Costs}_i \leftarrow \sum_{i=1}^{n} cost(t_i \in \Sigma_i)$ ;       /* where $cost(t) = |var|$ */
8  $(\sum_i)_{i=1}^n \leftarrow OrderByCost((\sum_i)_{i=1}^n, (\text{Costs})_{i=1}^n)$
   /* Step 2: Query Planning                                                                 */
9  $\overrightarrow{QP} \leftarrow \{\}$ ;                   /* query plan sequence */
10 for each $\sum_i \in (\sum_i)_{i=1}^n$ do                 /* for each PJG */
11     for each $\sum_j \in (\sum_i)_{i=i+1}^n$ do           /* for each succeeding PJG */
12         $\bowtie_{var} \leftarrow findJoinVariables(\sum_i, \sum_j)$ ;   /* find join variables */
13         if $\bowtie_{var} \neq \emptyset$ then            /* join exists */
14             $leftjoin(\bowtie_l \leftarrow \sum_i)$ AND $rightjoin(\bowtie_r \leftarrow \sum_j)$ ; /* build left and right joins arguments */
15             $\overrightarrow{QP} \leftarrow \overrightarrow{QP} \cup \{\bowtie_l, \bowtie_r\}$
   /* Step 3: Query Execution                                                                */
16 $\bowtie_{id} \leftarrow getLeafJoin(\overrightarrow{QP})$ ;      /* start from leaf joins */
17 while $\bowtie_{id} \neq \emptyset$ do
18     $\bowtie_{l,r} \leftarrow getJoin(\bowtie_{id})$ ;    /* get join arguments (left and right) */
19     $\bowtie_{ql} \leftarrow Translate.Query(\bowtie_{l,r}, \text{md})$ ;   /* translate joins to native query languages */
20     $\bowtie_{rdf} \leftarrow Translate.Result([\![\bowtie_{ql}]\!], \text{md})$ ;   /* translate query results to RDF */
21     $\bowtie_{l,r} \leftarrow Join.Result(\bowtie_{rdf})$ ;   /* join result set to left and right arguments */
22     $\mathcal{RS} \leftarrow \mathcal{RS} \cup \{\bowtie_{l,r}\}$ ;    /* update result set */
23     $\bowtie_{id}{++}$ ;                                  /* next join */
24 return $\mathcal{RS}$ ;                                   /* return result set */
```

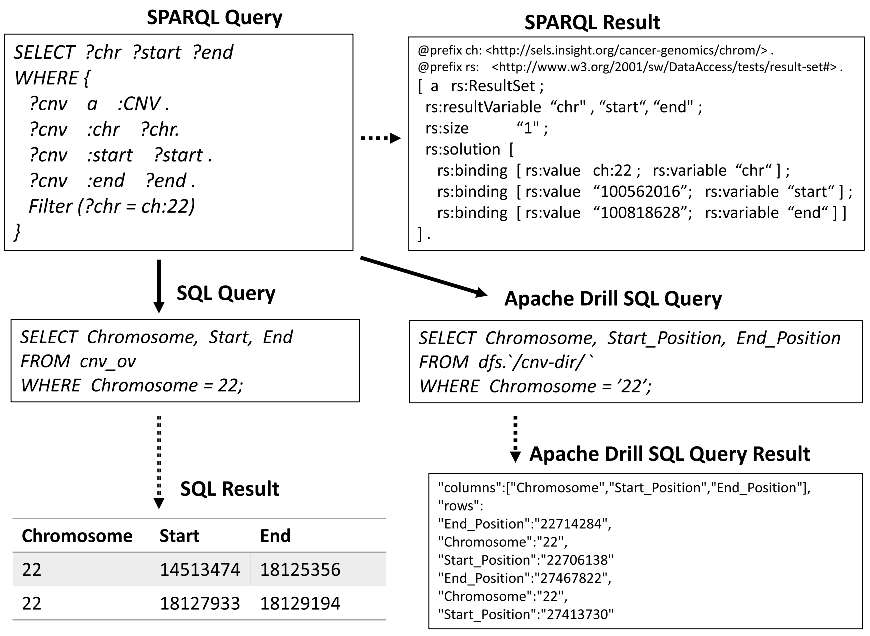
shown in the Fig. 4. The transformation between SQL and SPARQL queries is a non-trivial task, due to the difference in semantics between them. We used the query transformation method suggested here [48] that exploits R2RML mapping definitions.

The SQL result set obtained from RDB sources are transformed back to the standard SPARQL result format; (ii) **SPARQL to Apache Drill SQL**: In case of a CSV data source the SPARQL query is translated to the SQL format used by Apache Drill.[7] The different result sets obtained are transformed from its native format and aggregated in the RDF format, as shown in Listing 7. In the Fig. 4, it is important to note that alignment between matching variables/terminologies ("chr' corresponds to "Chromosome", "start" corresponds to "Start_Position") are resolved in the R2RML- /RML mappings defining ($\mathcal{M}$), which is input to the *Translate*.*Query*(.) and *Join*.*Result*(.) functions.

[7]https://drill.apache.org/docs/sql-reference/

E. POLYWEB CORRECTNESS

The goal of Algorithms 2 and 3 is to ensure that all possible answers for each BGP in the input (i.e., the answers possible for each BGP over a local merge of all data in $\mathcal{D}$) can be generated by joining the union of the results of individual triple patterns evaluated over the sources selected for those patterns. More formally, let $\mathfrak{D}$ denote the result of merging all data sets in $\mathcal{D}$ into a single global data set, let $bgp = \{t_1, \ldots, t_n\}$ denote a BGP, and let $[\![\mathfrak{D}]\!]_{bgp}$ denote the result of evaluating bgp over $\mathfrak{D}$ [46]. Next let R denote the sources selected for bgp by Algorithm 2, let $R(t)$ denote a data set selected for the triple pattern t in R, let $RS(E_{R(t)})$ denote data set R and triple pattern t selected by the Algorithm 3 for a predicate-based join group E, and let $[\![RS(E)]\!]_t$ denote the evaluation of triple pattern t over that predicate-based



```
SELECT ?chr ?start ?end
WHERE {
  ?cnv  a  :CNV .
  ?cnv  :chr  ?chr.
  ?cnv  :start  ?start .
  ?cnv  :end  ?end .
  Filter (?chr = ch:22)
}
```

```
@prefix ch: <http://sels.insight.org/cancer-genomics/chrom/> .
@prefix rs:   <http://www.w3.org/2001/sw/DataAccess/tests/result-set#> .
[ a  rs:ResultSet ;
 rs:resultVariable  "chr" , "start", "end" ;
 rs:size         "1" ;
 rs:solution  [
   rs:binding  [ rs:value  ch:22 ;  rs:variable  "chr" ] ;
   rs:binding  [ rs:value  "100562016";  rs:variable  "start" ] ;
   rs:binding  [ rs:value  "100818628";  rs:variable  "end" ] ]
] .
```

```
SELECT Chromosome, Start, End
FROM cnv_ov
WHERE Chromosome = 22;
```

```
SELECT Chromosome, Start_Position, End_Position
FROM dfs.`/cnv-dir/`
WHERE Chromosome = '22';
```

Chromosome	Start	End
22	14513474	18125356
22	18127933	18129194

```
"columns":["Chromosome","Start_Position","End_Position"],
"rows":
"End_Position":"22714284",
"Chromosome":"22",
"Start_Position":"22706138"
"End_Position":"27467822",
"Chromosome":"22",
"Start_Position":"27413730"
```

FIGURE 4. An example of query translations and results: SPARQL to SQL and Apache Drill SQL.

```
PREFIX ch: <http://examp.ly/chrom/>
PREFIX rs: <http://www.w3.org/[...]result-set#>

[ a rs:ResultSet;
rs:resultVariable "chr", "star", "end";
rs:size 3;
rdf:solution [
rs:binding
[ rs:value ch:22; rs:variable "chr" ],
[ rs:value "100562016"; rs:variable "start" ],
[ rs:value "100818628"; rs:variable "end" ]
], [
rs:binding
[ rs:value ch:22; rs:variable "chr" ],
[ rs:value "14513474"; rs:variable "start" ],
[ rs:value "18125356"; rs:variable "end" ]
], [
rs:binding
[ rs:value ch:22; rs:variable "chr" ],
[ rs:value "22706138"; rs:variable "start" ],
[ rs:value "22714284"; rs:variable "end" ]
]
].
```

Listing. 7. Aggregated Query Result in RDF.

join. The correctness condition we wish to satisfy is then as follows: $[\![\mathfrak{D}]\!]_{bgp} = [\![RS(E_1)]\!]_{t_1} \bowtie \ldots \bowtie [\![RS(E_n)]\!]_{t_n}$.

Let us start with a base case where all predicates are bound. In this case, the algorithm will select all data sets with matching predicates. In this case, it is not difficult to show that $[\![RS(E)]\!]_t = [\![\mathfrak{D}]\!]_t$ for all $t \in bgp$, and thus we have that $[\![\mathfrak{D}]\!]_{bgp} = [\![\mathfrak{D}]\!]_{t_1} \bowtie \ldots \bowtie [\![\mathfrak{D}]\!]_{t_n}$, which is the definition of the evaluation of BGPs [46]. Likewise, if we consider cases where some predicates are not bound, again the ASK queries will filter only irrelevant data sets, where it is again not difficult to show that $[\![RS(E)]\!]_t = [\![\mathfrak{D}]\!]_t$ for all $t \in bgp$ in RDF type data sets. However, the source selection algorithm is unable to probe the non-rdf data sets with ASK equivalent semantic, which is a limitation that makes both the algorithms incomplete in case of the unbound predicates. Second we must highlight that a known and non-trivial obstacle to completeness in federated scenarios is presented by blank nodes [49]; hence, per the motivating example, we assume that no blank nodes are used in the data.

F. POLYWEB COMPLEXITY

We analyses worst-case asymptotic runtime complexity for both the algorithms. For the source selection algorithm (Algorithm 2), we will consider q to be the size of the query encoding the number of triple patterns in the union of BGP; note that with this factor, we can abstract away the presence of multiple BGPs, the number of predicate in the query, etc., since these are bounded by q. Likewise we will consider d to be the number of data sets available and by p the number of unique predicates in all data sets. For each data set, we must check that a predicate is contained in that data set. This is feasible by a Merge sort over all predicates in a bgp with all predicates in the data set, resulting in $O(p \log(p))$ complexity for a given data set; and for all data sets we can more coarsely (but concisely) represent the complexity as $O(qd((q+p)\log(q+p)))$.

Algorithm 2 performs ASK queries – for unbound predicates – to each RDF type data set for each triple pattern matching the given criteria (bounded by q). In the general case, resolving ASK queries is NP-complete in combined complexity (considering both the size of the query and the data), even in the case that the query only contains a BGP and no features like optional and filters. However, since we only issue ASK queries with one triple pattern, and since the arity of the triple pattern is bounded, this step is feasible in (at least) time linear in the size of the data (for example, running a simple scan over all data), and so for all patterns, we have a resulting worst-case complexity of $O(qt)$ from this part of the source selection algorithm, where t is the number of triple patterns in the BGP. Hence we can merge the above factors to give a worst-case complexity of $O(qd(q+t)\log(q+p))$.

In the case of the QOPE algorithm (Algorithm 3), creating the predicate-based join groups from relevant data sets R and associated triple patterns for those groups can be done by first sorting the sets of data sets, (e.g., first by cardinality then by lexicographic order of the data set IRI), which is feasible in $O(t\log(t))$ in the worst-case with, e.g., a Merge sort implementation, then building the predicate-based joins amounts to linearly going through the sorted list. Then, finding join variables between predicate-based join groups (PJGs) can be achieved in quadratic time $O(|E|^2)$ in the size of PJGs. Once a set of join variables are identified between the PJGs, a query plan is constructed, which is delegated to the query execution component.

V. EVALUATION

In our Motivating Scenario section II, we introduced three core research questions:

- How can we devise source selection in a Web like federated scenario where data sources comply with different data models?
- How can we implement a web-based query federation plan that retrieves correct and complete results from different data models?
- Can we optimize and execute the query federation plan for different native data models in a manner that allows us to compete with off-the-shelf RDF query federation engines?

In terms of the first question, we have proposed the PolyWeb engine, which performs source selection based on the capabilities of native data sources and can deal with the multiplicity of data models while executing queries against those data sources; however, we have yet to see how efficient this new form of source selection performs when executing federated queries. In terms of the second question, we have proposed a query optimisation, planning and execution (QOPE) mechanism that retrieves correct query results in the presence of bound predicates (in a BGP) against different data models. We have yet to see how this optimisation compares to existing engines and completeness of the query result. For the third question, we have applied the PolyWeb engine in two experimental settings: (i) first, PolyWeb is applied on the mix of native data models (CSV, RDB, RDF) and the two RDF query federation (FedX, HiBISCuS) engines evaluated on the same three sets of data represented in the RDF format. We selected two approaches for comparison with PolyWeb, one index free approach, i.e., FedX and one index based approach, i.e., HiBISCuS, as both are the better performing engines compared to other SPARQL query federation engines [50]; (ii) second, PolyWeb (called PolyWeb-RDF) is applied on the same three RDF data sets to understand the gain or shortcomings compared to when PolyWeb executes in the PolyStore (CSV, RDB, RDF) environment.

Along these lines, in this section, we present the results of our evaluation comparing PolyWeb with two existing SPARQL query federation engines (FedX and HiBISCuS) and when PolyWeb applied only on RDF data sets (PolyWeb-RDF) for a variety of queries along a series of metrics and aspects.

A. EXPERIMENTAL SETUP

This section describes the experimental setup (i.e., data sets, setting, queries, and metrics) for evaluation of PolyWeb. The experimental material discussed in the following section and an implementation of PolyWeb are available online at the PolyWeb home page.[8]

a: DATA SETS

A total of three data sets, represented using different data models, are selected for experimental evaluation of PolyWeb, i.e. (i) COSMIC-CNV in RDF format; (ii) TCGA-OV-CNV in RDB format; and (iii) CNVD-CNV in CSV format, provided by COSMIC,[9] TCGA[10] and CNVD[11] data providers. These data sets are used in the studies conducted within the BIOOPENER project. In order to compare PolyWeb with the single data model query federation approaches (such as FedX and HiBISCuS), all the three data sets are transformed into the RDF format. An overview of the experimental datasets is given in Table 4, for instance, COSMIC-CNV consists of 37 million triples (6.54 GB size in RDF format); with equivalent 29 million database records. The "Raw Type" column represents the raw data format and the "No record" column presents the total number of records (rows) in each raw data type. The last column represents the cost ("RDFisation Time") associated with the transformation of raw data to RDF format. PolyWeb can avoid the data conversion cost (i.e., total 3 hours for 3.5 GB raw data). One of the main aims of PolyWeb is to avoid or reduce the data conversion cost in a federated environment.

b: SETTING

RDF data sets are loaded into different Virtuoso (Open Source v.7.2.4.2) SPARQL endpoints on separate physical machines. The relational data set is loaded into a MySQL database and the CSV dataset into Apache Drill. All experiments are carried out on a local network, so that network cost remains negligible. The machines used for experiments have a 2.60 GHz Core i7 processor, 8 GB of RAM and 500 GB hard disk running a 64-bit Windows 7 OS. The database con-

[8]http://polyweb.insight-centre.org/
[9]http://cancer.sanger.ac.uk/cosmic
[10]https://cancergenome.nih.gov/
[11]http://202.97.205.78/CNVD/

TABLE 4. Overview of experimental data sets.

Data set	№ trip	№ sub	№ pred	№ obj	RDF Data	Raw Type	Raw Data	№ record	RDFisation Time
COSMIC-CNV	37 M	1 M	18	0.3 M	6.54 GB	CSV	3.2 GB	29 M	3 Hours
TCGA-OV-CNV	10 M	1.8 M	6	1.2 M	494 MB	RDB	212 MB	2.6 M	2 Minutes
CNVD-CNV	1.7 M	0.2 M	12	1.7 M	128 MB	TSV	34 MB	0.2 M	3 Seconds
Total	49 M	2 M	36	3 M	7 GB	-	3.5 GB	32 M	3 Hours (approx)

TABLE 5. Summary of query characteristics.

Characteristics	QE-1	QE-2	QE-3	QE-4	QE-5	QE-6	QE-7	QE-8	QE-9	QE-10
№ of Patterns	5	2	5	5	5	9	6	2	5	5
Filters			✓	✓			✓		✓	
`LIMIT` modifier			✓	✓						
`DISTINCT` modifier		✓		✓	✓	✓	✓	✓	✓	

TABLE 6. Number of results returned for each query.

Systems	QE-1	QE-2	QE-3	QE-4	QE-5	QE-6	QE-7	QE-8	QE-9	QE-10
FedX	3	603	15	1	11	64	213	91	991	1090
HiBISCuS	3	603	15	1	11	64	213	91	991	1090
PolyWeb	3	603	15	1	11	64	213	91	991	1090
PolyWeb-RDF	3	603	15	1	11	64	213	91	991	1090

figuration parameters are described in the PolyWeb GitHub repository.[12]

c: QUERIES

We have designed 10 queries to evaluate and compare the performance of PolyWeb with FedX and HiBISCuS engines. Table 5 summaries the characteristics of these queries following similar dimensions to that used in the Berlin SPARQL benchmark [51], which shows that the queries have a variety of characteristics and complexity.

d: METRICS

We have measured six metrics for each query type to compare the performance of PolyWeb with other federation engines; (i) data summaries generation time and compression ratio; (ii) the number of sources selected; (iii) the number of SPARQL ASK requests; (iv) the average source selection time; (v) the number of results returned to assess result completeness relatively and (vi) the average query execution time.

B. EXPERIMENTAL RESULTS

In this section, we present the experimental results obtained for the given data sets, setting, queries and metrics discussed previously.

[12]https://github.com/yasarkhangithub/PolyWeb

TABLE 7. Data summaries generation time and compression ratio.

Systems	Time	Size	Ratio
PolyWeb	353 s	2 KB	99.999 %
HiBISCuS	936 s	80 KB	99.998 %
FedX	0 s	0 KB	100 %

1) DATA SUMMARIES GENERATION TIME AND COMPRESSION RATIO

The method of generating data summaries for PolyWeb is compared with various state-of-the-art approaches and the comparison results are shown in Table 7. The comparison metrics used are data summaries generation time **(time)**, data summaries size **(size)** and the data summaries reduction (**ratio**: computed as $1 - \frac{\text{index size}}{\text{total dump size}}$).

The results of Table 7 show that the different sizes of data summaries for all approaches are much smaller than the relative size of the RDF data dump. In the case of FedX, no data summaries are created since relevant sources are determined on-the-fly at runtime. PolyWeb produces the smallest data summaries by storing only the meta-data about predicates: for RDF datasets having a raw dump size of 7 GB, PolyWeb generates data summaries of size 2 KB, achieving 99.999% reduction. It should be noted however that although in relative terms HiBISCuS produces 40 times larger data summaries,

TABLE 8. Sum of triple-pattern-wise sources selected for each query.

System	QE-1	QE-2	QE-3	QE-4	QE-5	QE-6	QE-7	QE-8	QE-9	QE-10	Avg
PolyWeb	13	3	12	12	12	13	13	3	13	8	10
PolyWeb-RDF	13	3	12	12	12	13	13	3	13	8	10
FedX	12	3	11	11	12	13	13	3	13	8	10
HiBISCuS	12	3	11	11	12	13	13	3	13	8	10

TABLE 9. Number of SPARQL `ASK` requests used for source selection.

System	QE-1	QE-2	QE-3	QE-4	QE-5	QE-6	QE-7	QE-8	QE-9	QE-10	Avg
PolyWeb	0	0	0	0	0	0	0	0	0	0	0
PolyWeb-RDF	0	0	0	0	0	0	0	0	0	0	0
FedX	15	6	15	15	6	9	18	6	12	12	11
HiBISCuS	9	3	3	3	6	3	3	3	3	3	4

the absolute sizes of the data summaries are relatively small across both engines, where reduction rates remain above 99.9%.

Considering data summaries generation time, aside from FedX which incurs no data summaries generation costs, PolyWeb has achieved a significant performance gain over all the approaches. PolyWeb's data summaries generation time is 353 seconds as compared to 936 seconds for HiBISCuS. FedX incurs no data summaries generation cost but we suppose that PolyWeb's data summaries will reduce the load on remote endpoints and ultimately the overall query-execution time.

2) SELECTED SOURCES

The total number of triple pattern-wise (TP) sources selected by PolyWeb, PolyWeb-RDF, FedX and HiBISCuS for all the 10 queries are shown in Table 8. The average number of TP sources selected by each approach across all queries are depicted by the last column. FedX performs source selection using `ASK` queries for each triple pattern to find out which sources can actually answer an individual triple pattern. However, these sources might not contribute to the end results after performing a join between two triple patterns, i.e., after performing the join, results from some sources might be excluded. HiBISCuS, on the other hand, is a hybrid source selection approach, which uses both `ASK` queries and data summaries to identify the relevant sources. PolyWeb and PolyWeb-RDF return the same number of relevant sources.

The results show that, on average, PolyWeb, FedX and HiBISCuS identify the same number of sources (i.e., 10). The results in Table 8 show that in some queries PolyWeb overestimate the set of sources that contribute to the final query results. This is due to the reason that PolyWeb only considers the predicate specified in the triple pattern while FedX and HiBISCuS, along with predicate, also consider subject and object specified in the triple pattern. There can be cases where a data set covers the predicate but does not cover the bound object or subject or both specified in the triple pattern. These cases are pruned by FedX, e.g., QE-4 query.

3) NUMBER OF SPARQL ASK REQUESTS

The total number of SPARQL `ASK` requests sent to perform source selection for each query are shown in Table 9. FedX, as an index-free approach, performs SPARQL `ASK` requests at runtime during source selection for each triple pattern in query: hence FedX must run many more `ASK` queries than data summaries assisted engines. HiBISCuS is a hybrid approach that utilizes pre-computed data summaries as well as SPARQL `ASK` requests at runtime during source selection for each triple pattern in a query. Hence HiBISCuS greatly reduces the number of `ASK` queries used during source selection, by exploiting data summaries. On the other hand, PolyWeb uses data summaries for source selection, reverting to SPARQL `ASK` requests only when there is an unbound predicate in a triple pattern. The index-free approaches are flexible in scenarios where underlying sources are frequently updated, but it can incur a large cost in terms of SPARQL `ASK` requests used for source selection, which can in turn increase overall query execution time.

4) SOURCE SELECTION TIME

The comparison of source selection time for PolyWeb, PolyWeb-RDF, FedX and HiBISCuS for each query is shown in Fig. 5, where the y-axis is presented in log-scale. The rightmost set of bars compares the average source selection time over all queries. The indexes of HiBISCuS and PolyWeb remain quite small relative to total sizes of data and hence can easily be loaded into memory, where lookups can be performed in milliseconds. In the case of PolyWeb and PolyWeb-RDF, source selection is performed in less than a millisecond for all queries. On the other hand, remote `ASK` queries executions are orders of magnitude more costly in comparison with in-memory lookups. Hence we see that the source selection time for PolyWeb is much lower due to the

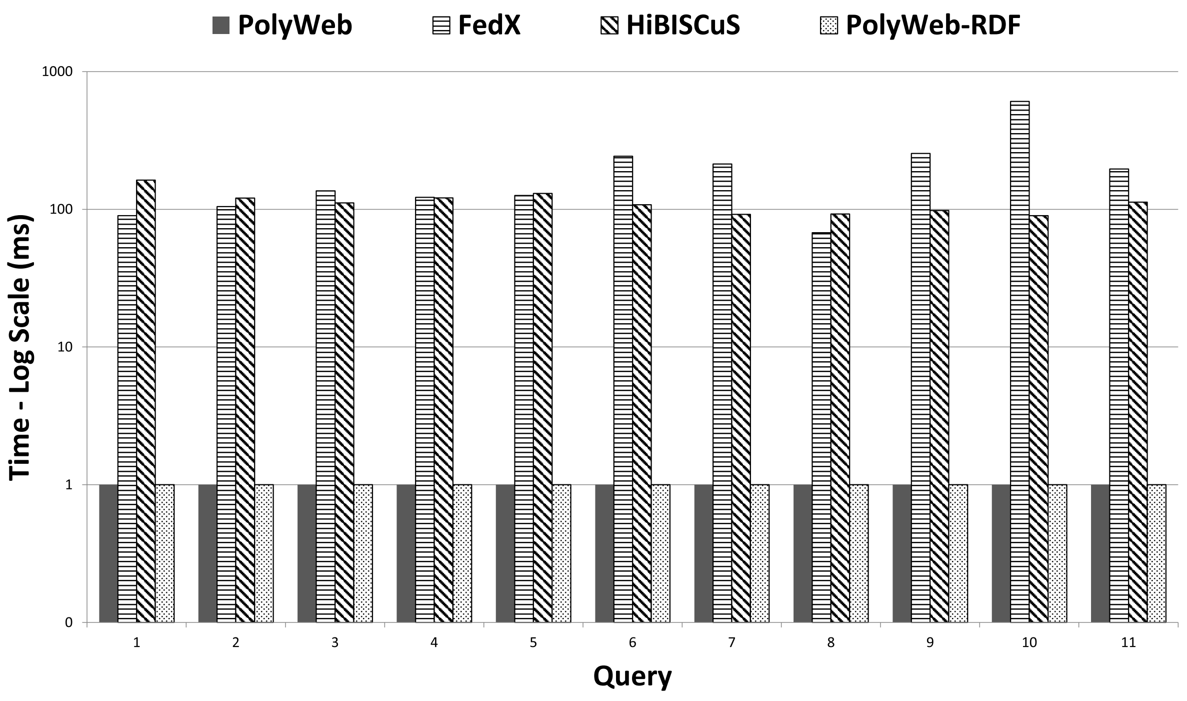


FIGURE 5. Comparison of source selection time.

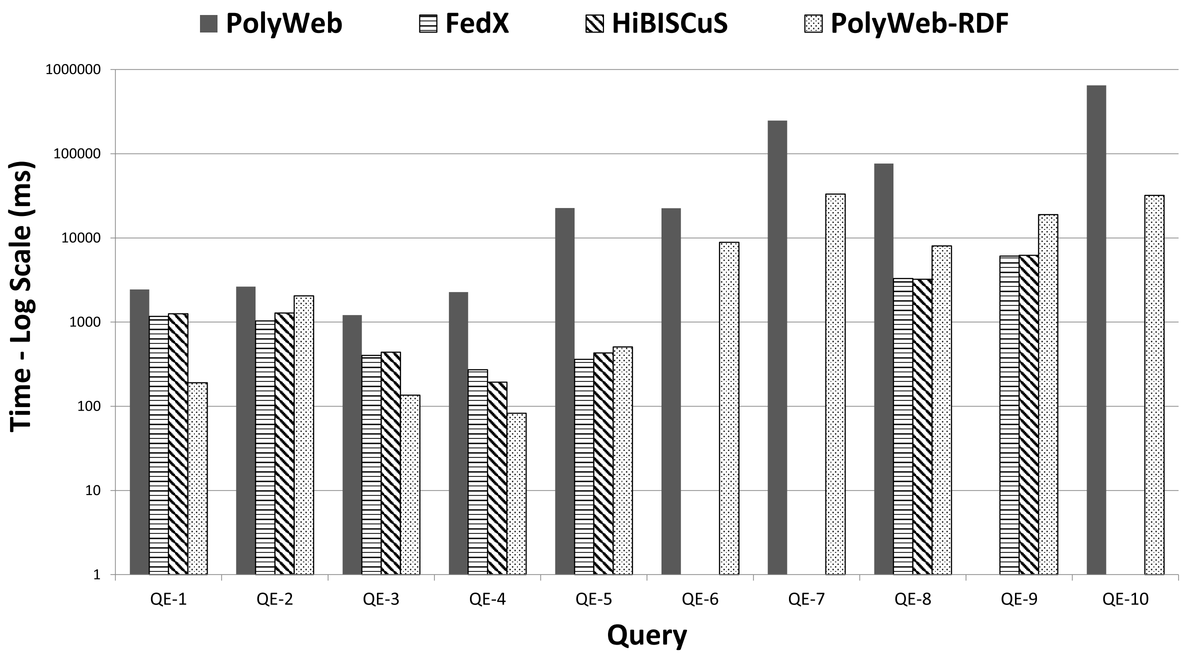


FIGURE 6. Comparison of query execution time.

reason that PolyWeb only uses ASK queries infrequently in general, as previously discussed. HiBISCuS sometimes relies on ASK queries for source selection along with index lookup.

5) RESULT COMPLETENESS

This indicates the query results retrieved from the federated data sets. All the three engines returned the same number of results corresponding to the ten queries (Table 6). FedX and HiBISCuS are designed to retrieve complete result sets and therefore, it implies that PolyWeb is capable of retrieving complete set of query results from the native data models.

6) QUERY EXECUTION TIME

For each query, a mean query execution time was calculated for PolyWeb, PolyWeb-RDF, FedX, and HiBISCuS by running each query ten times. Fig. 6 compares the overall mean query execution times, where the y-axis is log-scale. A time-out of 30 minutes on query execution; with these settings, FedX and HiBISCuS time-out in the case of three queries. On the other hand, PolyWeb times-out in one case and PolyWeb-RDF in none. Looking at query response times, apart from PolyWeb-RDF, FedX outperforms the other engines in most of the queries but in complex queries where FedX times-out, i.e., **QE-6**, **QE-7** and **QE-10**, PolyWeb outperforms it. The reason behind FedX time-out is the number of intermediate results to be joined locally. PolyWeb uses a block size of one while performing nested loop bind join which results in more remote requests while performing the join and hence more response time, but still, the results are comparable considering the additional factors which PolyWeb has to cope with.

The overall query execution time of a query federation engine can be influenced by various factors, such as join type, join order selection, block and buffer size, number of remote requests, intermediate results etc. PolyWeb focuses on querying heterogeneous data models in a federated setting which requires some additional costs apart from the mentioned factors. For example, queries are translated between different query languages (SPARQL, SQL, Apache Drill SQL) on different data models and most importantly, the RDF transformation of heterogeneous query results retrieved from different sources and data sets. PolyWeb performance is still comparable, knowing that additional run-time transformation cost is added. The cost of additional factors can be seen from the query execution time (Fig. 6) of PolyWeb-RDF, which is lower than PolyWeb in all the 10 queries. At the same time, PolyWeb uses optimisation factors as well, such as the predicate-based join group (PJG) that lessens the remote requests and local joins, which are the main culprits in query performance degradation in a federated setting.

Consider the case of example query (**QE-6**), where PolyWeb is way above the mark compared to FedX, making the additional costs, involved with Polystores, negligible. There are still many ways of improvements left which will be targeted in the near future to further make it to the mark, such as using dynamic cost model, parallel execution of joins and block nested bind joins (increasing block size).

VI. CONCLUSION & FUTURE WORKS

The work presented in this article is motivated in particular by the needs of the BIOOPENER project which aims at linking data across the large-scale cancer and biomedical repositories. PolyWeb's key approach is to query the vast data resources from their native data models and delegate querying load to specialized data storage engines. PolyWeb aims at reducing the expensive data conversion cost – loading curated data from multiple data models and sources to a centralized data warehouse for querying – while still being able to retrieve complete results from different native models. PolyWeb contributes on two major processes of a query federation (i) source selection: mechanism to find prospective data sets that can satisfy a given query; and (ii) query optimisation, planning, and execution: a mechanism to optimize a given query and devise an efficient plan to execute over different native data models.

PolyWeb currently exploits R2RML and RML in the query federation process but can be extended to allow other mapping languages to cover an open set of data models. Our evaluation results show that PolyWeb retrieves complete results set when compared with FedX and HiBISCuS federation engines. At the same time, PolyWeb can avoid the data conversion cost (i.e., total 3 hours for 3.5 GB raw data). While some queries reveal the overhead that the PolyWeb approach incurs, in some cases, it significantly outperforms the single data model query federation engines in terms of source and overall query execution time. This shows that in spite of the relatively basic optimizations we propose, the approach is viable already.

In our future work, there are a number of possible aspects to investigate with respect to enhancing the PolyWeb engine: (i) in addition to the basic SPARQL constructs (BGP, OPTIONAL, FILTER), we plan to implement features (e.g., support of blank nodes, variables on the predicate position) and SPARQL constructs (UNION, GROUP BY, ORDER BY etc.) often encountered in complex querying scenarios; (ii) we plan to devise a mechanism to probe non-RDF data sets at run-time in the presence of unbound predicates; (iii) the transformation between query results (non-RDF and RDF data sets) is a non-trivial task and we plan to optimize such transformations in a more declarative fashion, such as proposed in [42]; and finally (iv) we plan to include a dynamic cost model that will further optimize the PolyWeb engine.

ACKNOWLEDGMENT

This article is based on the previous work [52] of the authors that discussed the preliminary version of PolyWeb.

REFERENCES

[1] M. Stonebraker and U. Çetintemel, "'One size fits all': An idea whose time has come and gone," in *Proc. 21st Int. Conf. Data Eng. (ICDE)*, Tokoyo, Japan, Apr. 2005, pp. 2–11.

[2] O. Badawi *et al.*, "Making big data useful for health care: A summary of the inaugural MIT critical data conference," *JMIR Med. Inform.*, vol. 2, no. 2, p. e22, Aug. 2014.

[3] M. Schmachtenberg, C. Bizer, and H. Paulheim, "Adoption of the linked data best practices in different topical domains," in *The Semantic Web—ISWC* (Lecture Notes in Computer Science), vol. 8796, P. Mika, T. Tudorache, A. Bernstein, C. A. Welty, C. A. Knoblock, D. Vrandecic, P. T. Groth, N. F. Noy, K. Janowicz, and C. A. Goble, Eds. Riva del Garda, Italy: Springer, Oct. 2014, pp. 245–260.

[4] Z. D. Stephens *et al.*, "Big data: Astronomical or genomical?" *PLoS Biol.*, vol. 13, no. 7, p. e1002195, 2015.

[5] J. Duggan *et al.*, "The BigDAWG polystore system," *SIGMOD Rec.*, vol. 44, no. 2, pp. 11–16, 2015.

[6] A. J. Elmore *et al.*, "A demonstration of the BigDAWG polystore system," *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1908–1911, 2015.

[7] A. Jha *et al.*, "Towards precision medicine: Discovering novel gynecological cancer biomarkers and pathways using linked data," *J. Biomed. Semantics*, vol. 8, no. 1, pp. 40:1–40:16, 2017.

[8] Y. Khan *et al.*, "SAFE: SPARQL federation over RDF data cubes with access control," *J. Biomed. Semantics*, vol. 8, no. 1, pp. 5:1–5:22, 2017.

[9] M. Saleem, Y. Khan, A. Hasnain, I. Ermilov, and A.-C. N. Ngomo, "A fine-grained evaluation of SPARQL endpoint federation systems," *Semantic Web J.*, vol. 7, no. 5, pp. 493–518, 2016.

[10] M. Stonebraker *et al.*, "Mariposa: A wide-area distributed database system," *Very Large Data Base J.*, vol. 5, no. 1, pp. 48–63, 1996.

[11] D. J. DeWitt *et al.*, "Split query processing in polybase," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, K. A. Ross, D. Srivastava, and D. Papadias, Eds. New York, NY, USA: ACM, Jun. 2013, pp. 1255–1266.

[12] 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, 1990.

[13] G. Wiederhold, "Mediators in the architecture of future information systems," *IEEE Comput.*, vol. 25, no. 3, pp. 38–49, Mar. 1992.

[14] L. M. Haas, D. Kossmann, E. L. Wimmers, and Y. Yang, "Optimizing queries across diverse data sources," in *Proc. 23rd Int. Conf. Very Large Data Bases (VLDB)*, M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, Eds. Athens, Greece: Morgan Kaufmann, Aug. 1997, pp. 276–285.

[15] M. Lenzerini, "Data integration: A theoretical perspective," in *Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. Princ. Database Syst.*, L. Popa, S. Abiteboul, and P. G. Kolaitis, Eds. Madison, WI, USA: ACM, Jun. 2002, pp. 233–246.

[16] T. G. Mattson, V. Gadepally, Z. She, A. Dziedzic, and J. Parkhurst, "Demonstrating the BigDAWG polystore system for ocean metagenomics analysis," in *Proc. 9th Biennial Conf. Innov. Data Syst. Res. (CIDR)*, D. J. DeWitt, D. Kossmann, A. Ailamaki, and M. Stonebraker, Eds. Chaminade, CA, USA, Jan. 2017, pp. 1–9. [Online]. Available: www.cidrdb.org

[17] A. E. Johnson *et al.*, "MIMIC-III, a freely accessible critical care database," *Nature*, vol. 3, May 2016, Art. no. 160035.

[18] P. Sadalage and M. Fowler, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. London, U.K.: Pearson, 2012.

[19] F. Bugiotti, D. Bursztyn, A. Deutsch, I. Ileana, and I. Manolescu, "Invisible glue: Scalable self-tuning multi-stores," in *Proc. 7th Biennial Conf. Innov. Data Syst. Res. (CIDR)*, D. J. DeWitt, D. Kossmann, M. Balzinska, and M. Stonebraker, Eds. Asilomar, CA, USA, Jan. 2015, pp. 1–8. [Online]. Available: www.cidrdb.org

[20] E. Kharlamov *et al.*, "A semantic approach to polystores," in *Proc. IEEE Int. Conf. Big Data (BigData)*, J. Joshi, G. Karypis, L. Liu, X. Hu, R. Ak, Y. Xia, L. Xu, A. Sato, S. Rachuri, L. H. Ungar, P. S. Yu, R. Govindaraju, and T. Suzumura, Eds. Washington, DC, USA, Dec. 2016, pp. 2565–2573.

[21] E. Begoli, D. Kistler, and J. Bates, "Towards a heterogeneous, polystore-like data architecture for the US Department of Veteran Affairs (VA) enterprise analytics," in *Proc. IEEE Int. Conf. Big Data (BigData)*, J. Joshi, G. Karypis, L. Liu, X. Hu, R. Ak, Y. Xia, L. Xu, A. Sato, S. Rachuri, L. H. Ungar, P. S. Yu, R. Govindaraju, and T. Suzumura, Eds. Washington, DC, USA, Dec. 2016, pp. 2550–2554.

[22] S. Dasgupta, K. Coakley, and A. Gupta, "Analytics-driven data ingestion and derivation in the AWESOME polystore," in *Proc. IEEE Int. Conf. Big Data (BigData)*, J. Joshi, G. Karypis, L. Liu, X. Hu, R. Ak, Y. Xia, L. Xu, A. Sato, S. Rachuri, L. H. Ungar, P. S. Yu, R. Govindaraju, and T. Suzumura, Eds. Washington, DC, USA, Dec. 2016, pp. 2555–2564.

[23] S. J. Lynden, I. Kojima, A. Matono, and Y. Tanimura, "ADERIS: An adaptive query processor for joining federated SPARQL endpoints," in *On the Move to Meaningful Internet Systems—OTM* (Lecture Notes in Computer Science), vol. 7045, R. Meersman, T. S. Dillon, P. Herrero, A. Kumar, M. Reichert, L. Qing, B. C. Ooi, E. Damiani, D. C. Schmidt, J. White, M. Hauswirth, P. Hitzler, and M. K. Mohania, Eds. Hersonissos, Greece: Springer, Oct. 2011, pp. 808–817.

[24] M. Acosta, M. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus, "ANAPSID: An adaptive query processing engine for SPARQL endpoints," in *The Semantic Web—ISWC* (Lecture Notes in Computer Science), vol. 7031, L. Aroyo, C. A. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. F. Noy, and E. Blomqvist, Eds. Bonn, Germany: Springer, Oct. 2011, pp. 18–34.

[25] C. Basca and A. Bernstein, "Avalanche: Putting the spirit of the Web back into semantic Web querying," in *Proc. Posters Demonstrations Track, Collected Abstr. (ISWC)*, vol. 658, A. Polleres and H. Chen, Eds. Shanghai, China, Oct. 2010, pp. 1–4. [Online]. Available: http://ceur-ws.org/Vol-1035/iswc2013_poster_14.pdf

[26] B. Quilitz and U. Leser, "Querying distributed RDF data sources with SPARQL," in *The Semantic Web: Research and Applications*, vol. 5021, S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, Eds. Canary Islands, Spain: Springer, Jun. 2008, pp. 524–538.

[27] M. Saleem, A.-C. N. Ngomo, J. X. Parreira, H. F. Deus, and M. Hauswirth, "DAW: Duplicate-AWare federated query processing over the Web of data," in *The Semantic Web—ISWC* (Lecture Notes in Computer Science), vol. 8218, H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. A. Welty, and K. Janowicz, Eds. Sydney, NSW, Australia: Springer, Oct. 2013, pp. 574–590.

[28] A. Nikolov, A. Schwarte, and C. Hütter, "FedSearch: Efficiently combining structured queries and full-text search in a SPARQL federation," in *The Semantic Web—ISWC* (Lecture Notes in Computer Science), vol. 8218, H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. A. Welty, and K. Janowicz, Eds. Sydney, NSW, Australia: Springer, Oct. 2013, pp. 427–443.

[29] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and R. A. Schmidt, "FedX: A federation layer for distributed query processing on linked open data," in *The Semantic Web: Research and Applications* (Lecture Notes in Computer Science), vol. 6644, G. Antoniou, M. Grobelnik, E. P. B. Simperl, B. Parsia, D. Plexousakis, P. D. Leenheer, and J. Z. Pan, Eds. Heraklion, Greece: Springer, May 2011, pp. 481–486.

[30] X. Wang, T. Tiropanis, and H. C. Davis, "LHD: Optimising linked data query processing using parallelisation," in *Proc. Workshop Linked Data Web (WWW)*, vol. 996, C. Bizer, T. Heath, T. Berners-Lee, M. Hausenblas, and S. Auer, Eds. Rio de Janeiro, Brazil, May 2013. [Online]. Available: http://ceur-ws.org/Vol-996/papers/ldow2013-paper-06.pdf

[31] O. Görlitz and S. Staab, "SPLENDID: SPARQL endpoint federation exploiting VOID descriptions," in *Proc. 2nd Int. Workshop Consuming Linked Data (COLD)*, vol. 782, O. Hartig, A. Harth, and J. F. Sequeda, Eds. Bonn, Germany, Apr. 2011. [Online]. Available: http://ceur-ws.org/Vol-996/papers/GoerlitzAndStaab_COLD2011.pdf

[32] M. Saleem and A.-C. N. Ngomo, "HiBISCuS: Hypergraph-based source selection for SPARQL endpoint federation," in *The Semantic Web: Trends and Challenges* (Lecture Notes in Computer Science), vol. 8465, V. Presutti, C. d'Amato, F. Gandon, M. d'Aquin, S. Staab, and A. Tordia, Eds. Anissaras, Greece: Springer, May 2014, pp. 176–191.

[33] G. Montoya, H. SkafMolli, P. Molli, and M. Vidal, "Federated SPARQL queries processing with replicated fragments," in *The Semantic Web—ISWC* (Lecture Notes in Computer Science), vol. 9366, M. Arenas, Ó. Corcho, E. P. B. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. T. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, and S. Staab, Eds. Bethlehem, PA, USA: Springer, Oct. 2015, pp. 36–51.

[34] M. Rodríguez-Aguilar, R. Kontchakov, and M. Zakharyaschev, "Ontology-based data access: Ontop of databases," in *The Semantic Web—ISWC* (Lecture Notes in Computer Science), vol. 8218, H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. A. Welty, and K. Janowicz, Eds. Sydney, NSW, Australia: Springer, Oct. 2013, pp. 558–573.

[35] M. Bienvenu, B. ten Cate, C. Lutz, and F. Wolter, "Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP," in *Proc. 32nd ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst. (PODS)*, R. Hull and W. Fan, Eds. New York, NY, USA: ACM, Jun. 2013, pp. 213–224.

[36] S. Auer *et al.*, "The bigdataeurope platform—Supporting the variety dimension of big data," in *Web Engineering*, vol. 10360, Rome, Italy: Springer, 2017, pp. 41–59.

[37] V. Gadepally *et al.*, "The BigDAWG polystore system and architecture," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Waltham, MA, USA, Sep. 2016, pp. 1–6.

[38] S. Harris and A. Seaborne. SPARQL 1.1 Query Language, W3C Recommendation 21 Mar. 2013, World Wide Web Consortium, W3C Recommendation, Accessed: Mar. 21, 2013. [Online]. Available: http://www.w3.org/TR/2013/REC-sparql11-query-20130321/

[39] S. Das, S. Sundara, and R. Cyganiak. R2RML: RDB to RDF Mapping Language, World Wide Web Consortium, W3C Recommendation, Accessed: Sep. 27, 2012. [Online]. Available: http://www.w3.org/TR/2012/REC-r2rml-20120927/

[40] A. Dimou, M. V. Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. V. de Walle, "RML: A generic language for integrated RDF mappings of heterogeneous data," in *Proc. Workshop Linked Data Web*, vol. 1184, C. Bizer, T. Heath, S. Auer, and T. Berners-Lee, Eds. Seoul, South Korea, Apr. 2014, pp. 1–5. [Online]. Available: http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf

[41] A. Polleres, T. Krennwallner, N. Lopes, J. Kopecký, and S. Decker. XSPARQL Language Specification, World Wide Web Consortium, W3C Member Submission, Jan. 20, 2009. [Online]. Available: http://www.w3.org/Submission/2009/SUBM-xsparql-language-specification-20090120/

[42] M. Lefrançois, A. Zimmermann, and N. Bakerally, "A SPARQL Extension for Generating RDF from Heterogeneous Formats," in *The Semantic Web* (Lecture Notes in Computer Science), vol. 10249, E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler, and O. Hartig, Eds. Portorož, Slovenia: Springer, May 2017, pp. 35–50.

[43] M. Kay. XSL Transformations (XSLT) Version 3.0 W3C Recommendation 8 Jun. 2017, World Wide Web Consortium, W3C Recommendation, Jun. 8, 2017. [Online]. Available: http://www.w3.org/TR/2017/REC-xslt-20170608/
[44] M. Sporny, G. Kellogg, and M. Lanthaler. JSON-LD 1.0, A JSON-based Serialization for Linked Data, W3C Recommendation 16 Jan. 2014, World Wide Web Consortium, Jan. 16, 2014. [Online]. Available: http://www.w3.org/TR/2014/REC-json-ld-20140116/
[45] J. Tennison and G. Kellogg. Metadata Vocabulary for Tabular Data, W3C Recommendation 17 Dec. 2015, World Wide Web Consortium, W3C Recommendation, Dec. 17, 2015. [Online]. Available: http://www.w3.org/TR/2015/REC-tabular-metadata-20151217/
[46] J. Pérez, M. Arenas, and C. Gutiérrez, "Semantics and complexity of SPARQL," *ACM Trans. Database Syst.*, vol. 34, no. 3, pp. 16:1–16:45, 2009.
[47] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds, "SPARQL basic graph pattern optimization using selectivity estimation," in *Proc. 17th Int. Conf. World Wide Web (WWW)*, J. Huai, R. Chen, H. Hon, Y. Liu, W. Ma, A. Tomkins, and X. Zhang, Eds. Beijing, China: ACM, Apr. 2008, pp. 595–604.
[48] M. Rodríguez-Aguilar and M. Rezk, "Efficient SPARQL-to-SQL with R2RML mappings," *J. Web Semantics*, vol. 33, pp. 141–169, Aug. 2015.
[49] A. Stolpe and J. Halvorsen, "Distributed query processing in the presence of blank nodes," *J. Web Semantics*, vol. 8, no. 6, pp. 1001–1021, 2017.
[50] M. Saleem, A. Hasnain, and A.-C. N. Ngomo, 'LargeRDFBench: A billion triples benchmark for SPARQL endpoint federation," *J. Web Semantics*, vol. 48, pp. 85–125, Jan. 2018.
[51] C. Bizer and A. Schultz, "The Berlin SPARQL benchmark," *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 2, pp. 1–24, 2009.
[52] Y. Khan, A. Zimmermann, A. Jha, D. Rebholz-Schuhmann, and R. Sahay, "Querying Web polystores," in *Proc. IEEE Int. Conf. Big Data (BigData)*, J. Nie, Z. Obradovic, T. Suzumura, R. Ghosh, R. Nambiar, C. Wang, H. Zang, R. A. Baeza-Yates, X. Hu, J. Kepner, A. Cuzzocrea, J. Tang, and M. Toyoda, Eds. Boston, MA, USA, Dec. 2017, pp. 3190–3195.

YASAR KHAN received the bachelor's degree in computer science from the University of Peshawar, Pakistan, in 2007, and the M.S. degree in information technology from the National University of Sciences and Technology, Pakistan, in 2011. He is currently pursuing the Ph.D. degree with the Insight Centre for Data Analytics, National University of Ireland Galway. His Ph.D. work focuses on query federation over heterogeneous data sources and query optimization. He is also a Research Assistant with the Insight Centre for Data Analytics, National University of Ireland Galway.

ANTOINE ZIMMERMANN graduated from the École Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble. He received the M.Sc. degree and the Ph.D. degree in computer science from Université Joseph Fourier, Grenoble, France, in 2008. From 2009 to 2010, he was with the Digital Enterprise Research Institute (Insight NUI Galway), National University of Ireland, Galway. From 2010 to 2011, he was a Lecturer with the Institut Nationale de Sciences Appliquées de Lyon and a Researcher with the Laboratoire d'InfoRmatique en Image et Systéme d'information. Since 2011, he has been teaching at the École des Mines de Saint-Étienne and is a member of the Institut Henri Fayol. He conducts his research at the Laboratoire Hubert Curien as a part of the research group Connected Intelligence.

ALOKKUMAR JHA is currently pursuing the Ph.D. degree with the Insight Centre for Data Analytics, National University of Ireland Galway. His primary work focuses on biomedical data analytics, graph pattern mining, and machine learning predictions. He was with various organizations, such as Harvard University, Tata Cancer Hospital, UC Davis, and SAP Labs, where he has covered a range of areas, such as databases, graph theory, pattern mining, and deep learning algorithms. He has served as a member of the HL7 Group for Genomics and Clinical Data Integration.

VIJAY GADEPALLY received the B.Tech. degree in electrical engineering from IIT Kanpur and the M.Sc. and Ph.D. degrees in electrical and computer engineering from The Ohio State University. He has served as the President of the Council of Graduate Students and the Chairperson of the Student Commercialization Board, The Ohio State University. He is currently a Senior Scientist with the Massachusetts Institute of Technology (MIT) Lincoln Laboratory. He is also with the Computer Science and Artificial Intelligence Laboratory. He has also been the National Director of Employment Concerns for the National Association of Graduate-Professional Students. In 2011, he received the Outstanding Graduate Student Award at The Ohio State University. In 2017, he received the Early Career Technical Achievement Award at the MIT Lincoln Laboratory and was named to AFCEA's inaugural "Under 40" list.

MATHIEU D'AQUIN was a Senior Research Fellow with the Knowledge Media Institute, Open University, where he led the Data Science Group. He is currently a Professor of informatics, specialized in data analytics and semantic technologies, with the Insight Centre for Data Analytics, National University of Ireland Galway. He is leading research and development activities around the meaningful sharing and exploitation of distributed information. He has worked on applying the technologies coming out of his research, especially semantic web/linked data technologies, in various domains, including medicine, education, especially through learning analytics, smart cities, the Internet of Things, and personal data management.

RATNESH SAHAY received the bachelor's degree in information technology from the University of Southern Queensland, Australia, in 2002, the M.S. degree in distributed systems from the KTH Royal Institute of Technology, Sweden, in 2006, and the Ph.D. degree in computer science with a specialization in healthcare informatics from the National University of Ireland, Galway, in 2012. He has over 12 years of working/research experience in the healthcare and life sciences domain. He is currently the Head of Semantics at the Insight Centre for Data Analytics, eHealth and Life Sciences Research Group, National University of Ireland Galway. He has been active and leading the European and National R&D projects with an emphasis on e-health and semantic interoperability. He was a member of the OASIS SEE Technical Committee. He is a member of the W3C OWL 2, W3C HCLS, and HL7 working groups.