More Complete Resultset Retrieval from Large Heterogeneous RDF Sources

André Valdestilhas*
AKSW Research Group
University of Leipzig
Germany
valdestilhas@informatik.uni-leipzig.de

Tommaso Soru
AKSW Research Group
University of Leipzig
Germany
tsoru@informatik.uni-leipzig.de

Muhammad Saleem
AKSW Research Group
University of Leipzig
Germany
saleem@informatik.uni-leipzig.de

ABSTRACT

Over the last years, the Web of Data has grown significantly. Various interfaces such as LOD Stats, LOD Laudromat, SPARQL endpoints provide access to the hundered of thousands of RDF datasets, representing billions of facts. These datasets are available in different formats such as raw data dumps and HDT files or directly accessible via SPARQL endpoints. Querying such large amount of distributed data is particularly challenging and many of these datasets cannot be directly queried using the SPARQL query language. In order to tackle these problems, we present WimuQ, an integrated query engine to execute SPARQL queries and retrieve results from large amount of heterogeneous RDF data sources. Presently, WimuQ is able to execute both federated and non-federated SPARQL queries over a total of 668,166 datasets from LOD Stats and LOD Laudromat as well as 559 active SPARQL endpoints. These data sources represent a total of 221.7 billion triples from more than 5 terabytes of information from datasets retrieved using the service "Where is My URI" (WIMU). Our evaluation on state-of-the-art real-data benchmarks shows that WimuQ retrieves more complete results for the benchmark queries.

CCS CONCEPTS

• **Information systems → Extraction, transformation and loading**.

KEYWORDS

Data Integration; Data Interlinking; Link Traversal based SPARQL query; SPARQL federated query; Dataset discovery; resultset coverage

ACM Reference Format:
André Valdestilhas, Tommaso Soru, and Muhammad Saleem. 2019. More Complete Resultset Retrieval from Large Heterogeneous RDF Sources. In *Proceedings of the 10th International Conference on Knowledge Capture (K-CAP '19), November 19–21, 2019, Marina Del Rey, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3360901.3364436

*All authors contributed equally.

K-CAP '19, November 19–21, 2019, Marina Del Rey, CA, USA

ACM ISBN 978-1-4503-7008-0/19/11…$15.00
https://doi.org/10.1145/3360901.3364436

1 INTRODUCTION

Currently, the LOD Laudromat along with LOD stats have more than 250 billion facts which are available on 668,166 datasets and 559 active SPARQL endpoints [40]. Querying such large amount of distributed data is particularly challenging in which Federated query processing is one of the key components for accessing this collection of information through these data sources. In general, there are two types of available approaches to execute federated SPARQL queries over distributed RDF data sources [30]. (1) `Query federation over multiple SPARQL endpoints (QFME)` which collects distributed information from multiple SPARQL endpoints. Commonly, the list of SPARQL endpoints is provided as an input to the federation engine. The federation engine decomposes the original query into multiple sub-queries and execute them accordingly to a specific query execution plan. (2) `Link Traversal based SPARQL federation (LTSF)` which uses the URI lookups to collect information from distributed RDF data sources. Such approaches do not force the data providers to publish their data as SPARQL endpoints. Instead, the only requirement is that the RDF data sources must follow the Linked Data principles[1] [14].

However, a particular shortcoming of the QFME approaches is that many of the RDF datasets in Linking Open Data (LOD) are not publicly available via SPARQL endpoints. Beek et al. [5] has shown that around 90% of the information published are available only as data dumps. On the other hand, the LTSF approaches are unable to perform lookups for non-dereferenceable URIs. Hence, they may retrieve empty or incomplete results for the latter type of approaches. Note that our previous work [40] shows that around 43% of the URIs of more than 660K RDF datasets from LODStats [8] and LODLaundromat [5] are non-dereferenceable.

In this work, dubbed WimuQ, we provide an approach that overcomes the limitations of the state of the art regarding to the coverage of the results. WimuQ is a hybrid (endpoints federation + link traversal) federated SPARQL query processing engine which collects distributed information from SPARQL endpoints as well as raw data dumps and HDT files [10]. WimuQ integrates exciting state-of-the-art LTSF and QFME approaches into a single query execution framework. Additionally, WimuQ overcomes the problem of non-dereferenceable URIs by making use of our previous index of 4.2 billion URIs from more than 660k RDF datasets [40]. Our evaluation on state-of-the-art real-data benchmarks shows that WimuQ retrieves more complete results on three different benchmarks for federated SPARQL query engines [29, 31, 36].

The contributions of this paper are as follows:

[1] http://www.w3.org/DesignIssues/LinkedData.html

- We present a hybrid SPARQL query processing engine which is is able to retrieve results from 559 active SPARQL endpoints (with a total of 163.23 billion triples) and 668,166 datasets (with a total of 58.49 billion triples) from LOD Stats and LOD Laudromat. Thus, to the best of our knowledge, WimuQ is the first federated SPARQL query processing engine that executes SPARQL queries over a net total of 221.7 billion triples
- As part of WimuQ, we present a low-cost API dubbed *SPARQL-a-lot* to execute SPARQL queries over LOD-a-lot [9], a 300 GB HDT file of the LOD cloud. For the first time, to the best of our knowledge, we make use of the Concise Bounded Descriptions (CBDs)footnote 6 to execute federate SPARQL queries.
- We make use of the WIMU index [40] to intelligently select the capable (also called relevant) [32] data sources pertaining to the given SPARQL query. We evaluated our integrated query engine on two real-data federated SPARQL querying benchmarks *LargeRDFBench* [28],*FedBench* [36] and one non-federated real data SPARQL benchmark selected from *FEASIBLE* [31] benchmarks generation framework.

WimuQ is open source and available from https://github.com/firmao/wimuT along with complete evaluation results. The web version of the WimuQ is available from https://w3id.org/wimuq/.

The rest of the paper is organized as follows: Section 2 introduces the related work, followed by WimuQ approach in Section 4. Section 5 shows the evaluation results and discussion. Finally, Section 6 concludes the paper along with future directions.

2 RELATED WORK

In this section we provide a brief overview of approaches for federated query processing.

Query federation over multiple SPARQL endpoints: DARQ [27], ADERIS [20], FedX [37], Lusail [1], SPLENDID [12], CostFed [35], ANAPSID [2], SemaGrow [6], Odyssey [24], KBox [21] and MULDER [7] are examples of state-of-the-art SPARQL endpoint federation engines. These approaches can be further divided into 3 categories namely *index-only*, *index-free*, and hybrid (index+ASK) approaches [30].

DARQ, ADERIS and KBox are examples of the index-only SPARQL query federation approaches. KBox perform federation query processing over distributed RDF indices while DARQ and ADERIS operate federation query over multiple SPARQL endpoints. DARQ implements a cardinality-based query planner with bind join implementation in nested loops. ADERIS is an adaptive query engine that implements a cost-based query planner. It also makes use of the index-based nested loop join.

FedX and Lusail are examples of the index-free query federation approaches over multiple SPARQL endpoints. FedX only makes use of ASK queries for source selection. It implements a heuristic-based query planner. Comparing to DARQ, the number of endpoints requests is greatly reduced by using bind joins in a block nested loop fashion [37]. A query rewriting algorithm is used to push computation to the local endpoints by relying on information about the underlying RDF datasets. It implements a selectivity-aware query execution plan generation.

SPLENDID, CostFed, ANAPSID, SemaGrow, and Odyssey are examples of the hybrid (index+ASK) SPARQL query federation approaches over multiple SPARQL endpoints. SPLENDID performs cost-based optimization using VOID statistics from datasets. It makes use of bind and hash joins [30]. CostFed also implements a cost-based query planner. The source selection is closely related to HiBISCuS [32]. Both bind and symmetric hash joins are used for data integration. ANAPSID [2] is an adaptive query federation engine that adapts its query execution at runtime according to the data availability and condition of the SPARQL endpoints. ANAPSID implements adaptive group and adaptive dependent joins [30]. SemaGrow adapts source selection approach from SPLENDID. It performs a cost-based query planning based on VOID statistics about datasets. SemaGrow implements bind, hash, and merge joins. Odyssey is also a cost-based federation engine. MULDER describes data sources in terms of RDF molecule templates and utilize these template for source selection, and query decomposition and optimization.

DAW [33] and Fedra [25] are examples of duplicate-aware query federation approaches over multiple SPARQL endpoints.

SaGe[23] is a stateless preemptable SPARQL query engine for public endpoints. The system makes use of preemptable query plans and time-sharing scheduling, SaGe tackles the problem of RDF data availability for complex queries in public endpoints. Consequently, SaGe provides a convenient alternative to the current practice of copying RDF data dumps.

Link Traversal based SPARQL federation: LDQPS [18], SIHJoin [19], WoDQA [3], and SQUIN [15] are examples of traversal-based federated SPARQL query processing approaches. Both LQPS and SIHJoin make use of the index and online discovery via link-traversal to identify the relevant sources pertaining to the given SPARQL query. They implement symmetric hash join. WoDQA performs hybrid (index+ASK) source selection approach. It implements nested loop and bind joins. SQUIN discovers the potentially relevant data during the query execution and thus produce incremental query results. SQUIN uses a heuristic for query execution plan generation, adapted from [13]. As a physical implementation of the logical plans, SQUIN uses a synchronized pipeline of iterators such that the i-th operator is responsible for the i-th triple pattern of the given SPARQL query. More recent studies [17] investigated 14 different approaches to rank traversal steps and achieve a variety of traversal strategies. A more exhaustive survey of the traversal-based SPARQ query federation is provided in [16].

Beside the above query federation strategies, low-cost triple pattern fragments (TPF) interfaces [41] can also be used to execute federated SPARQL queries. Comunica [39] is a highly modular meta engine for federated SPARQL query evaluation over support heterogeneous interfaces types, including self-descriptive Linked Data interfaces such as TPF. The system also enables querying over heterogeneous sources, such as SPARQL endpoints and data dumps in RDF serializations. The main drawback here is that the user should know in advance the dataset where the query will be executed.

One of the targets and motivation to build WimuQ was to discover which dataset the SPARQL query can be executed, unfortunately, with exception of SQUIN [15], none of the related works

provides this feature. Despite the fact that some of them incorporate Nquad support that allows the possibility to know the graph of the triple. The provenance system of WimuQ also includes dump files and endpoints with a rank provided by WIMU [40].

3 PRELIMINARIES

We now introduce the terminology and symbolism that is used throughout the rest of this paper[2].

RDF graph. An RDF graph is a set of RDF triples which has a set of subjects and objects of triples in the graph called nodes. Given an infinite set U of URIs, an infinite set B of blank nodes and an infinite set of literals L, a RDF triple is a triple $\langle s, p, o\rangle$ where the subject $s \in (U \cup B)$, the predicate $p \in U$ and the object $o \in (U \cup B \cup L)$. An RDF triple represents an assertion of a "piece of knowledge", so if the triple $\langle s, p, o\rangle$ exists, then, the logical assertion $p(s, o)$ holds true. An RDF graph is also represented by a collection of RDF triples and it can be seen as a set of statements describing, partially or completely, a certain knowledge.

Basic Graph Pattern [3] is a set of Triple Patterns[11].

RDF Dataset An RDF dataset is a set:
$G, (< u_1 >, G_1), (< u_2 >, G2), ...(< u_n >, G_n)$
where G and each G_i are graphs, and each $<u_i>$ is an IRI. Each $<u_i>$ is distinct. G is called the default graph. $(<u_i>, G_i)$ are called named graphs.

SPARQL is the language to query RDF datasets, in which the formal definition of a SPARQL Query is: A SPARQL Abstract Query is a tuple (E, D, R) where E is a SPARQL algebra expression, D is an RDF Dataset and R is a query form.

More formally, let **Q** be a SPARQL query and $\mathcal{D}$ be a finite set of the datasets which can be retrieved using WIMU [40]. For each data source D $\in \mathcal{D}$, we write G(D) to denote the underlying RDF graph exposed by D. When requesting data source D to execute a SPARQL query Q, we expect that the result of D is the set $[\![Q]\!]_{G(D)}$. Hence, we define the result set S(**Q**) of a SPARQL query over the federation $\mathcal{D}$ to be a set of solution mappings that is equivalent to $[\![Q]\!]_{G(\mathcal{D})}$ where $G_{\mathcal{D}} = \bigcup_{D \in \mathcal{D}} G(D)$.

We formalize the problem in two steps as follows: (1) *find the datasets on which a given query can be executed* and (2) *return the query results from the selected datasets.*[4]

4 THE WIMUQ

In this section, we explain the WimuQ approach to the SPARQL query processing. We assume that the reader is familiar with the concepts of RDF and SPARQL, including the notions of an RDF triple, a triple pattern, a basic graph pattern (BGP), and subject, predicate, object of the triple pattern.

Figure 1 shows the workflow of the query processing in WimuQ, which comprises of four main steps: (1) the user issue a SPARQL query to the WimuQ interface, which (2) extracts all the URIs used in the given user query. Note the URIs can be used in subject, predicate, or objects of the SPARQL triple patterns. (3) The extracted URIs are then searched in the WIMU index, which gives all the relevant datasets where the extracted URIs can be found. (4) The relevant datasets are furthered filtered based on the source selection algorithm and (5) the finally-selected relevant datasets are then queried by using the different query processors, depending on the type (HDT, endpoint, datadump, dereferenceable dataset) of the datasets; (6) the results generated by the different query processors are then integrated and sent back to the user. The whole process is like a black box to the end user: the user only sends the query and get back the results without knowing the underlying query execution steps.

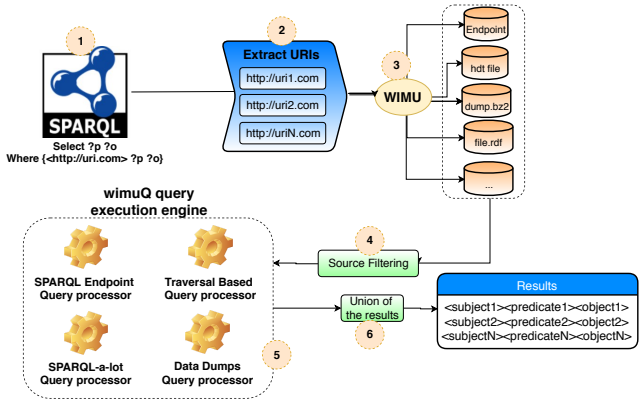

Figure 1: WimuQ's query processing workflow.

Now we go into the details of these steps, explaining the different components.

4.1 WimuQ source selection

One of the important optimization steps in federated SPARQL query processing is the *source selection* [26, 32]. The goal of the source selection is to identify the potentially relevant datasets (also called sources) to the given query. We make use of the WIMU service [40] to select the potentially relevant sources. The WIMU service[5] provides URIs lookup facility and identify those datasets which contain the given URI. Currently, this service processed more than 58 billion unique triples and indexed 4.2 billion URIs from more than 660k RDF datasets obtained from the most used Linked Data hubs including LODStats [4] and LOD Laundromat [5]. The identified WIMU datasets can be of four types: (1) SPARQL endpoint, (2) HDT file, (3) dataset with dereferenceable URIs, (4) data dump with non-dereferenceable URIs. The service is both available from a web interface as well as can be queried from a client application using the standard HTTP protocol.

The WimuQ source selection algorithm is given in Algorithm 1 which takes a SPARQL query Q as input. We provide the set of extracted URIs (from step 2) U to the WIMU service and retrieve the relevant datasets D pertaining to the given URIs (Lines 1-4 of Algorithm 1). Now for each relevant dataset $d \in D$, if the d is a SPARQL endpoint, we extract the individual triple patterns t of

[2]Further information about RDF and related definition can be found at: https://www.w3.org/TR/rdf-sparql-query/
[3]Basic Graph Patterns: https://www.w3.org/TR/rdf-sparql-query/#BasicGraphPatterns
[4]Throughout the paper, we refer to RDF dump files or any datasets accessible using a SPARQL endpoint as *datasets*, *data sources* or simply *sources*.
[5]WIMU URIs lookup service is available from: http://wimu.aksw.org/

the query and perform a SPARQL ASK of t in dataset d (Lines 5-8 of Algorithm 1). We add the dataset d into the set of relevant SPARQL endpoints E, if the SPARQL ASK query returns true (Lines 9-13 of Algorithm 1). We directly add d into sets H and T if d is an HDT file or if the d is a SPARQL endpoint or dataset with dereferenceable URIs, respectively (Lines 14-19 of Algorithm 1). Finally, if d is dataset with non-dereferenceable URIs, we apply the RDFSlice [22] technique to only get the relevant RDF slice of the dataset. The RDF slice is then considered as a relevant dataset (Lines 20-23 of Algorithm 1). The purpose of only adding the relevant slice of the complete dataset is to reduce the processing overhead, explained in the next section.

Algorithm 1 The WimuQ source selection

Input: Q ▷ The SPARQL query
Output: E, H, T, N ▷ Hash sets of relevant sources of SPARQL endpoints, HDT files, dataset with dereferenceable URIs, datadump with non-dereferenceable URIs, respectively

1: **U** = *extractURIs*(**Q**) ▷ Extract the URIs from the SPARQL query
2: **for all u** ∈ **U do**
3: **D** = **D** ∪ *wimuLookup*(**u**) ▷ Datasets from the WIMU
4: **end for**
5: **for all** d ∈ **D do** ▷ For each dataset
6: **if** d is SPARQL endpoint **then**
7: **for all** t ∈ **Q do** ▷ For each Triple pattern in query
8: b = *ASK*(t,d) ▷ SPARQL ASK of t in d
9: **if** b = true **then**
10: **E**.add(d)
11: **end if**
12: **end for**
13: **end if**
14: **if** d is HDT file **then**
15: **H**.add(d)
16: **end if**
17: **if** d is dataset with dereferenceable URIs **then**
18: **T**.add(d)
19: **end if**
20: **if** d is datadump with non-dereferenceable URIs **then**
21: **N**.add(*RDFSlice*(d)) ▷ only add the relevant slice
22: **end if**
23: **end for**
24: return **E, H, T, N** ▷ final relevant sources set

4.2 WimuQ query execution

We make use of the four – SPARQL endpoint, Link Traversal-based, SPARQL-a-alot, Data dumps – query processor to execute federated queries over the aforementioned four types of WIMU datasets. The relevant data sources for each of these query processors are returned by the WimuQ source selection discussed in previous section.

In WimuQ, we used FedX [37] query processor for SPARQL endpoints query federation and SQUIN for traversal-based query federation. The reason for choosing FedX for SPARQL endpoints federation and SQUIN for traversal-based federation is due the fact that they do not require any pre-computation of dataset statistics and hence are able to retrieve up-to-date results. Thus both are able to run federated queries with zero initial knowledge. In addition, both produce reasonably query runtime performances comparing to state-of-the-art approaches [15, 30, 35].

The list of required endpoints URLs for FedX are returned from the previously discussed source selection Algorithm (ref. **E** of Algorithm 1). The potentially relevant dereferenceable URIs data sources (ref. **T** of Algorithm 1) are already identified by the WimuQ sources selection algorithm. Thus, we reduced the search space by only considering the dereferenceable URIs data sources.

As mentioned before, FedX can only works with public SPARQL endpoints. SQUIN needs dereferenceable URIs. Both of these engines are unable to execute SPARQL queries over non-dereferenceable URIs datadumps: SPARQL endpoint federation approaches cannot execute queries over such datadumps as they are not exposed as SPARQL endpoints, link traversal-based approaches fail to retrieve results as the URIs are non-dereferenceable. We need to download the dumps first, load it locally, and run some query processing API (e.g., JENA or Sesame) on the loaded datasets. However, we can not simply download the complete dumps and process it locally due to their large amount of data. To solve this problem, we make use of the Wimu index to only select those data dumps which are potentially capable to execute the given query. These datadumps are further sliced by using the RDFSlice technique, to only select the required chunks of the datadumps which will provide results to the given SPARQL query. The identified chunks are finally loaded in a local Apache Jena model. The model is then use to execute federated queries.

Finally, we propose a query processor named *SPARQL-a-lot* to execute federated SPARQL queries over HDT files. The SPARQL-a-lot is a CBD-based[6] query execution described in Algorithm 2. The algorithm takes a SPARQL query Q as input and return the resultset O of the query execution over HDT file. First, the algorithm extracts all of the BGPs from Q (Line 2 of Algorithm 2). The next step is to extract the subjects, predicates, and objects of all the triple patterns used in Q (Line 3 of Algorithm 2). The CBDs are then generated from extracted subjects, predicates, and objects (Line 4 of Algorithm 2). The extracted CBDs constitute an RDF dataset of set of triples T which are then loaded into a local Jena model (Line 5 of Algorithm 2). The query Q is then finally executed over the Jena model and results are returned (Lines 6-7 of Algorithm 2). The duplicated results from different query processing engines is removed after collecting the final results.

The results generated by each of WimuQ's processors are finally integrated and sent back to the user.

4.3 Practical example

Given the following SPARQL query (LD1 from FEDBench[36]):

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * WHERE { ?paper
<http://data.semanticweb.org/ns/swc/ontology#isPartOf>
<http://data.semanticweb.org/conference/iswc/2008/
poster_demo_proceedings> .
?paper <http://swrc.ontoware.org/ontology#author> ?p .
?p rdfs:label ?n .
```

[6] https://www.w3.org/Submission/CBD/

Algorithm 2 Query execution on LOD-a-lot

Input: **Q** ▷ The SPARQL query
Output: **O** ▷ The results of the query

1: **procedure** SPARQL-A-LOT(**Q**)
2: **BGP** = *extractBGP*(**Q**) ▷ Extract the BGP from the SPARQL query
3: **Tbgp** = extractSPO(**BGPs**)
4: **T** = executeAPILOD-a-LOT(**Tbgp**) ▷ Generating CBDs from s, p, o
5: createTDBJena(**T**)
6: **O** = execSPARQLTDBJena(**Q**)
7: return **O**
8: **end procedure**

}

With WimuQ we were able to identify 4 HDT datasets available from https://tinyurl.com/WimuQexData. All of these datasets are from LOD Laundromat repository [5] containing information about the conference ISWC2008 [38], under the domain http://data.semanticweb.org, originally from SW Dog Food dataset[7]. Thus, *SPARQL-a-lot* is the only query processing engine (because of the selected datasets are HDTs) which is considered for executing the given SPARQL queries. The final results of the query execution is available from https://tinyurl.com/WimuQExample.

5 EVALUATION

In this section, we present the evaluation setup and the corresponding results that validate our hypothesis that we can improve the resultset retrieval if we automatically identify potentially relevant sources from heterogeneous RDF data even if the URIs are not dereferenceable anymore. The goal of this evaluation is to show that by combining different SPARQL query processing approaches, we are able to retrieve more complete results as compared to the results retrieved by the individual approaches.

5.1 Experimental setup

5.1.1 Benchmarks used: Since WimuQ aims to execute SPARQL queries over real-world RDF datasets, we chose three – FedBench [36], LargeRDFBench [28], Feasible [31] – real-world RDF datasets benchmarks in our evaluation:

- **FedBench** is federated SPARQL querying benchmarks. It comprises of a total of 25 queries and 9 real-world interconnected datasets. FedBench queries are further divided into three main categories: (1) 7 queries from Life Sciences (LS) domain, (2) 7 queries from Cross Domain (CD), and (3) 11 queries named Linked Data (LD) for link traversal-based approaches. The detailed statistics of the benchmark's datasets and queries are given in FedBench[36].
- **LargeRDFBench** is also a federated SPARQL querying benchmark. It comprises a total of 40 queries and 13 real-world interconnected datasets. FedBench queries are further divided into four main categories: (1) 14 *Simple* queries, (2) 10 *Complex* queries, (3) 8 *Large Data* queries, and (4) 8 *Complex+High Data Sources* queries. The detailed statistics of the benchmark's datasets and queries are given in [28].
- **FEASIBLE** is a benchmark generation framework which generates customized benchmarks for the queries logs. In our evaluation, we chose exactly the same benchmarks used in [31]: (1) 175 queries benchmark generated from DBpedia queries log and (2) 175 queries benchmark generated from Semantic Web Dog Food (SWDF) queries log. Further advanced statistics of the used datasets and queries can be found in [31].

To the best of our knowledge, these are the state-of-the-art from the real-data SPARQL benchmarks. All of the 415 queries used in our evaluation is publicly available[8].

5.1.2 Hardware: All the experiments were done on a modest machine with 200 GB of Hard Disk, 8 GB of RAM and a 2.70GHz single core processor. Each of the queries was run 5 times and the average of the results are presented.

5.1.3 SPARQL endpoints: As previously stated, the query federation over multiple SPARQL endpoints approaches requires the set of endpoint URLs to be provided as input to the federation engine. We chose a total of 539 active SPARQL endpoints available from LOD cloud[9]. We filtered the endpoints URLs[10] and the total number of triples hosted by each of these endpoints.

5.1.4 Metrics: Since WimuQ aims to retrieve more complete results within the reasonable amount of time, we choose two metrics: (1) coverage in terms of the number of results retrieved from the query executions and (2) the time taken to execute the benchmark queries.

5.1.5 Approaches: As mentioned in Section 2, different federation engines available to federate SPARQL queries over endpoints and traversal-based federation. We chose FedX [37] for SPARQL endpoint federation and SQUIN [15] for traversal-based query federation. The reason for choosing these two engines is due the fact they do not require any pre-computation of dataset statistics and hence able to retrieve up-to-date results and able to run federated queries with zero initial knowledge. In addition, both these engines perform reasonably well in terms of query runtime performances w.r.t state-of-the-art approaches [15, 30, 35]. For the sake of completeness, we also compared WimuQ with SPARQL-a-lot and WimuDumps.

5.2 Results

Coverage of the results: The main purpose of WimuQ is to devise a federation engine which is able to retrieve more complete results for the given SPARQL queries. Figure 2 shows a comparison of the selected approaches in terms of the average of the number of results retrieved for the different queries categories of the selected benchmarks. The complete results for individual queries can be found on our aforementioned project website. By using different query processing engines, our approach is able to retrieve more

[7] https://old.datahub.io/dataset/semantic-web-dog-food

[8] Queries available from https://github.com/firmao/wimuT/blob/master/queriesLocation.txt

[9] List of SPARQL endpoints: https://lod-cloud.net/lod-data.json

[10] Endpoints URLs with size: https://goo.gl/H2t5ko

results as compared to the results retrieved by only using SPARQL endpoints federation engine (i.e, FedX) link traversal engine (i.e., SQUIN), data dumps, or HDT files. In our evaluation, the average resultset size of WimuQ is 8481 across the three benchmarks. Out of these, WimuQ collects about 91% of the results from wimuDumps (avg. resultset size 7651), 7% from SPARQL endpoints (avg. resultset size 556), and 1% from SPARQL-a-lot (avg. resultset size 74).

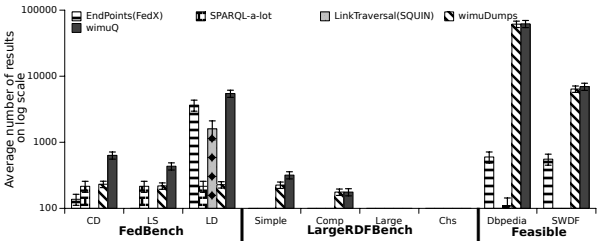

Figure 2: Average number of results retrieved by the selected approaches across different queries categories of the selected benchmarks.

For FedBench, the WimuQ avg. resultset size is 2,253. Out of these results, about 55% are collected from SPARQL endpoints by using FedX query processing engines (avg. resultset size 1,262). The LinkTraversal (SQUIN) contributed about 25% of the total results (avg. resultset size 549), wimuDumps contributed about 10% of the total results (avg. resultset size 226), and SPARQL-a-lot also contributed about 10% of the results (avg. resultset size 215).

For LargeRDFBench, the WimuQ avg. resultset size 123. Out of these results, about 81% are collected from wimuDumps (avg. resultset size 100). The SPARQL endpoints contributed about 14% of the results (avg. resultset size 17). The LinkTraversal(SQUIN) contributed 6% of the total results (avg. resultset size 6), and SPARQL-a-lot did not provide results (avg. resultset size 0).

For FEASIBLE, the WimuQ avg. resultset size 34,537. Out of these results, about 98% are collected from wimuDumps (avg. resultset size 33,893). The SPARQL endpoints contributed about 1.6% (avg. resultset size 577). The LinkTraversal(SQUIN) approach contributed only about 0.15% (avg. resultset size 54). Finally, SPARQL-a-lot query processing engine only retrieved about 0.03% results (avg. resultset size 11).

In summary, WimuQ is able to retrieve at least one resultset for 76% of the overall 415 queries. The results clearly shows that by combining different query processing engines into a single SPARQL query execution framework lead towards more complete resultset retrieval. An important observation is that the selected approaches are mostly not able to retrieve results for the Large Data and Complex+High (Ch) queries categories of LargeRDFBench. The reason is getting zero results for the Large Data queries is that these queries retrieve results from the LinkedTCGA [34] datasets which were not publicly available via SPARQL endpoints, were not indexed by WIMU, also were not reachable via link traversals. While Ch queries often require higher number of distributed datasets in order to compute the final resultset of the queries. Thus, the approaches were able to find all of the relevant datasets, required to compute the final resultset of the queries. The average number of datasets[11] queries by WimuQ for the selected benchmarks queries categories in given in Figure 3. We can clearly see the highest number of datasets are selected for Ch queries of LargeRDFBench.

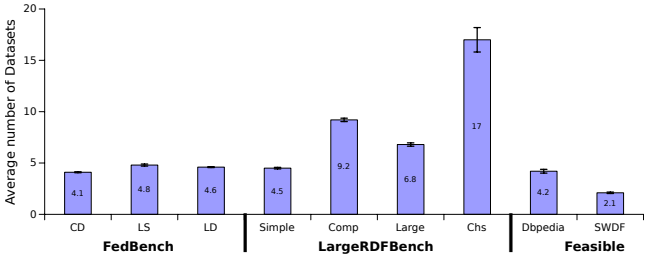

Figure 3: Average number of datasets discovered and queried by WimuQ across different queries categories of the selected benchmarks.

Query runtime performances: Figure 4 shows a comparison of the selected query processing engines in terms of the average query run times for the different queries categories of the selected benchmarks. The average query runtime of WimuQ is 17 minutes across the three benchmarks. The average query execution to collect results from wimuDumps is about 2 minutes, which in turn is followed by query execution over SPARQL endpoints (avg. query runtime 13 minutes), SPARQL-a-lot (avg. query runtime 58 seconds) and LinkTraversal (SQUIN) (avg. query runtime 36 seconds). Interestingly, WimuQ collects about 91% of the results from wimuDumps yet its average execution time is smaller than query execution over SPARQL endpoints which provide only 7% of the total results. One possible reason for this could be that in SPARQL endpoint federation, the query processing task split among multiple selected SPARQL endpoints and hence network and the number of intermediate results play an important role in the quer runtime performances.

For FedBench, the WimuQ average query runtime 20 minutes. Out of this, the average avg. query runtime over SPARQL endpoints is 16 minutes, followed by wimuDumps (avg. query runtime 2 minutes), LinkTraversal (SQUIN) (avg. query runtime 49 seconds), and SPARQL-a-lot (avg. query runtime 46 seconds), respectively. For LargeRDFBench, the average query execution of WimuQ is 11 minutes. Out of this query execution over SPARQL endpoints took 8 minuts on average, followed by wimuDumps (avg. query runtime 2 minutes), SPARQL-a-lot (avg. query runtime 35 seconds), and LinkTraversal (SQUIN) (avg. query runtime 25 seconds), respectively. For FEASIBLE, the WimuQ takes on average of 24 minutes per query execution. Out of this, the query federation over SPARQL endpoints took about 18 minutes on average. Which is followed by wimuDumps (avg. query runtime 3 minutes), SPARQL-a-lot (avg. query runtime 1), and LinkTraversal (SQUIN) (avg. query runtime 36 seconds), respectively.

As an overall query runtime evaluation, we can clearly see there is a trade-off between the recall and query runtimes: the highest the recall the highest the query runtimes. Finding a balance between the recall and runtime would be an interesting research question to be considered in the future.

[11]Here we also point the number of datasets discovered

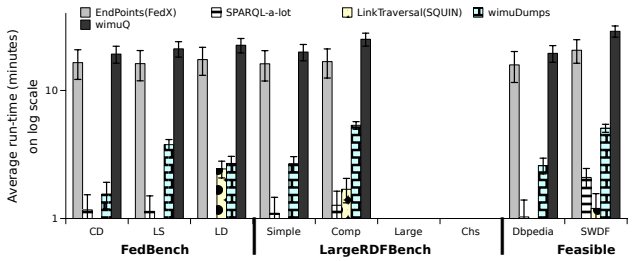

Figure 4: Average query runtimes of the selected approaches across different queries categories of the selected benchmarks.

The results of our evaluation lead us to validate our hypothesis that we can improve the resultset retrieval if we identify potentially relevant sources from heterogeneous RDF data.

6 CONCLUSIONS AND FUTURE WORK

We presented an approach to execute SPARQL queries over a large amount of heterogeneous RDF data sources available from different interfaces and in different formats. We made use of the Wimu service to identify the potentially relevant sources to the qiven SPARQL query. We discussed two main types of federated SPARQL query processing approaches namely the endpoints federation and traversal-based federation. The former type of federation only able to execute federated queries over the data available from SPARQL endpoint. While the later, faces problem of URI's dereferenceability. To overcome these issues we proposed a hybrid (endpoints+link-traversal-based) federation engines which integrates four different types of SPARQL query processing engines. Currently, WimuQ able to execute both federated and non-federated SPARQL queries over a total of 668k datasets available from LOD Stats, LOD Laudromat, and LOD cloud active SPARQL endpoints. We evaluated WimuQ by using three state-of-the-art real-data SPARQL benchmarks. We showed that WimuQ is able to successful execute (with some results) majority of the benchmark queries without any prior knowledge of the data sources. In addition, the WimuQ resultset recall is higher with reasonable query execution times.

As future, we will add more URIs into the the WIMU index in order to retrieve more complete and fast results. Furthermore, we will add TPF interfaces and query execution engines such Comunica [39] and SAGE [23] into the WimuQ query execution engine. We will continue maintaining[12] and developing, extending WimuQ to include the publicly-available triple pattern fragments interfaces as well[13].

ACKNOWLEDGMENTS

This work has been supported by the project LIMBO (Grant no. 19F2029I), OPAL (no. 19F2028A), KnowGraphs (no. 860801), and SOLIDE (no. 13N14456), CNPq Brazil under grants No. 201536/2014-5 and Deutsche Forschungsgemeinschaft (DFG) - Project-number: 317044652. Special thanks to Thomas Riechert.

[12]Sustainability plan: https://github.com/firmao/wimuT/wiki/Sustainability-plan

[13]In case of more information is needed we have an extended version of this paper available at http://139.18.13.76:8082/KCAP_extended.pdf

REFERENCES

[1] Ibrahim Abdelaziz, Essam Mansour, Mourad Ouzzani, Ashraf Aboulnaga, and Panos Kalnis. 2017. Lusail: a system for querying linked data at scale. *Proceedings of the VLDB Endowment* 11, 4 (2017), 485–498.

[2] Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. 2011. ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints. In *The Semantic Web – ISWC 2011*, Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Noy, and Eva Blomqvist (Eds.). Lecture Notes in Computer Science, Vol. 7031. Springer Berlin Heidelberg, 18–34. https://doi.org/10.1007/978-3-642-25073-6_2

[3] Ziya Akar, Tayfun Gökmen Halaç, Erdem Eser Ekinci, and Oguz Dikenelli. 2012. Querying the Web of Interlinked Datasets using VoID Descriptions.. In *C.Bizer et al., editors, Linked Data on the Web (LDOW2012) in CEUR Workshop Proceedings*, Vol. 937.

[4] Sören Auer, Jan Demter, Michael Martin, and Jens Lehmann. 2012. LODStats–an extensible framework for high-performance dataset analytics. In *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 353–362.

[5] Wouter Beek, Laurens Rietveld, Hamid R Bazoobandi, Jan Wielemaker, and Stefan Schlobach. 2014. LOD laundromat: a uniform way of publishing other people's dirty data. In *International Semantic Web Conference*. Springer, 213–228.

[6] Angelos Charalambidis, Antonis Troumpoukis, and Stasinos Konstantopoulos. 2015. SemaGrow: Optimizing Federated SPARQL Queries. In *Proceedings of the 11th International Conference on Semantic Systems (SEMANTICS '15)*. ACM, New York, NY, USA, 121–128. https://doi.org/10.1145/2814864.2814886

[7] Kemele M Endris, Mikhail Galkin, Ioanna Lytra, Mohamed Nadjib Mami, Maria-Esther Vidal, and Sören Auer. 2018. Querying Interlinked Data by Bridging RDF Molecule Templates. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXIX*. Springer, 1–42.

[8] Ivan Ermilov, Jens Lehmann, Michael Martin, and Sören Auer. 2016. LODStats: The data web census dataset. In *International Semantic Web Conference*. Springer, 38–46.

[9] Javier D Fernández, Wouter Beek, Miguel A Martínez-Prieto, and Mario Arias. 2017. LOD-a-lot: A queryable dump of the LOD cloud. (2017).

[10] Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. 2013. Binary RDF Representation for Publication and Exchange (HDT). *Web Semantics: Science, Services and Agents on the World Wide Web* 19, 0 (2013). http://www.websemanticsjournal.org/index.php/ps/article/view/328

[11] George HL Fletcher. 2008. An algebra for basic graph patterns. In *workshop on Logic in Databases, Rome, Italy*.

[12] Olaf Görlitz and Steffen Staab. 2011. SPLENDID: SPARQL Endpoint Federation Exploiting VoID Descriptions. In *O. Hartig, A. Harth, and J. F. Sequeda, editors, 2nd International Workshop on Consuming Linked Data (COLD 2011) in CEUR Workshop Proceedings*, Vol. 782.

[13] Olaf Hartig. 2011. Zero-Knowledge Query Planning for an Iterator Implementation of Link Traversal Based Query Execution. In *The Semantic Web: Research and Applications*, Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Pan (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 154–169.

[14] Olaf Hartig. 2013. An Overview on Execution Strategies for Linked Data Queries. In *Datenbank-Spektrum*, Vol. 13. Springer, 89–99.

[15] Olaf Hartig. 2013. SQUIN: a traversal based query execution system for the web of linked data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 1081–1084.

[16] Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. 2009. Executing SPARQL queries over the web of linked data. In *International Semantic Web Conference*. Springer, 293–309.

[17] Olaf Hartig and M Tamer Özsu. 2016. Walking without a map: Ranking-based traversal for querying linked data. In *International Semantic Web Conference*. Springer, 305–324.

[18] Günter Ladwig and Thanh Tran. 2010. Linked Data Query Processing Strategies. In *The Semantic Web – ISWC 2010*, PeterF. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, JeffZ. Pan, Ian Horrocks, and Birte Glimm (Eds.). Lecture Notes in Computer Science, Vol. 6496. Springer Berlin Heidelberg, 453–469. https://doi.org/10.1007/978-3-642-17746-0_29

[19] Günter Ladwig and Thanh Tran. 2011. SIHJoin: Querying Remote and Local Linked Data. In *The Semantic Web: Research and Applications*, Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Pan (Eds.). Lecture Notes in Computer Science, Vol. 6643. Springer Berlin Heidelberg, 139–153. https://doi.org/10.1007/978-3-642-21034-1_10

[20] Steven Lynden, Isao Kojima, Akiyoshi Matono, and Yusuke Tanimura. 2011. ADERIS: An Adaptive Query Processor for Joining Federated SPARQL Endpoints. In *R. Meersman, T. Dillon, P. Herrero, A. Kumar, M. Reichert, L. Qing, B.-C. Ooi, E. Damiani, D.C. Schmidt, J. White, M. Hauswirth, P. Hitzler, M. Mohania, editors, On the Move to Meaningful Internet Systems (OTM2011), Part II. LNCS*. Vol. 7045. Springer Heidelberg, 808–817.

[21] Edgard Marx, Ciro Baron, Tommaso Soru, and Sören Auer. 2017. KBox—Transparently Shifting Query Execution on Knowledge Graphs to the Edge. In *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*. IEEE, 125–132.

[22] Edgard Marx, Saeedeh Shekarpour, Tommaso Soru, Adrian MP Braşoveanu, Muhammad Saleem, Ciro Baron, Albert Weichselbraun, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, and Sören Auer. 2017. Torpedo: Improving the state-of-the-art rdf dataset slicing. In *Semantic Computing (ICSC), 2017 IEEE 11th International Conference On*. IEEE, 149–156.

[23] Thomas Minier, Hala Skaf-Molli, and Pascal Molli. 2018. SaGe: Preemptive Query Execution for High Data Availability on the Web. *CoRR* abs/1806.00227 (2018). arXiv:1806.00227 http://arxiv.org/abs/1806.00227

[24] Gabriela Montoya, Hala Skaf-Molli, and Katja Hose. 2017. The Odyssey approach for optimizing federated SPARQL queries. In *International Semantic Web Conference*. Springer, 471–489.

[25] Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. 2015. *ISWC*. Chapter Federated SPARQL Queries Processing with Replicated Fragments.

[26] Alexander Potocki, Muhammad Saleem, Tommaso Soru, Olaf Hartig, Martin Voigt, and Axel-Cyrille Ngonga Ngomo. 2017. Federated SPARQL Query Processing Via CostFed. (2017).

[27] Bastian Quilitz and Ulf Leser. 2008. Querying Distributed RDF Data Sources with SPARQL. In *The Semantic Web: Research and Applications*, Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis (Eds.). Lecture Notes in Computer Science, Vol. 5021. Springer Berlin Heidelberg, 524–538. https://doi.org/10.1007/978-3-540-68234-9_39

[28] Muhammad Saleem, Ali Hasnain, and Axel-Cyrille Ngonga Ngomo. 2016. LargeRDFBench: A Billion Triples Benchmark for SPARQL Endpoint Federation. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web* 0, 0 (2016). http://www.websemanticsjournal.org/index.php/ps/article/view/513

[29] Muhammad Saleem, Ali Hasnain, and Axel-Cyrille Ngonga Ngomo. 2018. Largerdfbench: a billion triples benchmark for sparql endpoint federation. *Journal of Web Semantics* 48 (2018), 85–125.

[30] Muhammad Saleem, Yasar Khan, Ali Hasnain, Ivan Ermilov, and Axel-Cyrille Ngonga Ngomo. 2015. A fine-grained evaluation of SPARQL endpoint federation systems. *Semantic Web Journal* (2015), 1–26. https://content.iospress.com/articles/semantic-web/sw186

[31] Muhammad Saleem, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. 2015. FEASIBLE: A Feature-Based SPARQL Benchmark Generation Framework. In *The Semantic Web-ISWC 2015*. Springer, 52–69. https://svn.aksw.org/papers/2015/ISWC_FEASIBLE/public.pdf

[32] Muhammad Saleem and Axel-Cyrille Ngonga Ngomo. 2014. HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation. In *The Semantic Web: Trends and Challenges*, Valentina Presutti, Claudia d'Amato, Fabien Gandon, Mathieu d'Aquin, Steffen Staab, and Anna Tordai (Eds.). Lecture Notes in Computer Science, Vol. 8465. Springer International Publishing, 176–191. https://doi.org/10.1007/978-3-319-07443-6_13

[33] Muhammad Saleem, Axel-Cyrille Ngonga Ngomo, Josiane Xavier Parreira, HelenaF. Deus, and Manfred Hauswirth. 2013. DAW: Duplicate-AWare Federated Query Processing over the Web of Data. In *The Semantic Web – ISWC 2013*, Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, JosianeXavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz (Eds.). Lecture Notes in Computer Science, Vol. 8218. Springer Berlin Heidelberg, 574–590. https://doi.org/10.1007/978-3-642-41335-3_36

[34] Muhammad Saleem, Shanmukha S. Padmanabhuni, Axel-Cyrille Ngonga Ngomo, Jonas S. Almeida, Stefan Decker, and Helena F. Deus. 2013. Linked Cancer Genome Atlas Database. In *M. Sabou, E. Blomqvist, T. Di Noia, H. Sack, T. Pellegrini, editors, Proceedings of the 9th International Conference on Semantic Systems*. ACM, New York, NY, USA, 129–134. https://doi.org/10.1145/2506182.2506200

[35] Muhammad Saleem, Alexander Potocki, Tommaso Soru, Olaf Hartig, and Axel-Cyrille Ngonga Ngomo. 2018. Costfed: Cost-based query optimization for sparql endpoint federation. *Semantics* 137 (2018), 163–174.

[36] Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. 2011. FedBench: A Benchmark Suite for Federated Semantic Data Query Processing. In *The Semantic Web – ISWC 2011*, Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Noy, and Eva Blomqvist (Eds.). Lecture Notes in Computer Science, Vol. 7031. Springer Berlin Heidelberg, 585–600. https://doi.org/10.1007/978-3-642-25073-6_37

[37] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. 2011. FedX: Optimization Techniques for Federated Query Processing on Linked Data. In *The Semantic Web – ISWC 2011*, Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Noy, and Eva Blomqvist (Eds.). Lecture Notes in Computer Science, Vol. 7031. Springer Berlin Heidelberg, 601–616. https://doi.org/10.1007/978-3-642-25073-6_38

[38] Amit P Sheth, Steffen Staab, Michael Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, and Krishnaprasad Thirunarayan. 2008. The Semantic Web-ISWC 2008. (2008).

[39] Ruben Taelman, Joachim Van Herwegen, Miel Vander Sande, and Ruben Verborgh. 2018. Comunica: a modular SPARQL query engine for the web. In *International Semantic Web Conference*. Springer, 239–255.

[40] Andre Valdestilhas, Tommaso Soru, Markus Nentwig, Edgard Marx, Muhammad Saleem, and Axel-Cyrille Ngonga Ngomo. 2018. Where is my URI?. In *European Semantic Web Conference*. Springer, 671–681.

[41] Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. 2016. Triple Pattern Fragments: a low-cost knowledge graph interface for the Web. *Web Semantics: Science, Services and Agents on the World Wide Web* 37 (2016), 184–206.