LargeRDFBench: A Billion Triples Benchmark for SPARQL Endpoint Federation

Muhammad Saleem[a,b,*], Ali Hasnain[c], Axel-Cyrille Ngonga Ngomo[a,b]

[a] *Universität Leipzig, IFI/AKSW, PO 100920, D-04009 Leipzig*
[b] *DICE, University of Paderborn, Germany*
[c] *Insight Centre for Data Analytics, National University of Ireland, Galway*

Abstract

Gathering information from the distributed Web of Data is commonly carried out by using SPARQL query federation approaches. However, the fitness of current SPARQL query federation approaches for real applications is difficult to evaluate with current benchmarks as they are either synthetic, too small in size and complexity or do not provide means for a fine-grained evaluation. We propose LargeRDFBench, a billion-triple benchmark for SPARQL query federation which encompasses real data as well as real queries pertaining to real bio-medical use cases. We evaluate state-of-the-art SPARQL endpoint federation approaches on this benchmark with respect to their query runtime, triple pattern-wise source selection, number of endpoints requests, and result completeness and correctness. Our evaluation results suggest that the performance of current SPARQL query federation systems on simple queries (in terms of total triple patterns, query result set sizes, execution time, use of SPARQL features etc.) does not reflect the systems' performance on more complex queries. Moreover, current federation systems seem unable to deal with real queries that involve processing large intermediate result sets or lead to large result sets.

Keywords: Benchmark, SPARQL, Federated Queries, Linked Data, RDF

1. Introduction

Accessing the distributed compendium that is the Web of Data is most commonly carried out by using SPARQL queries. In particular, federated SPARQL queries are used when data from several sources is required to satisfy the needs of the user. The importance of SPARQL queries for Linked Data management has led to the development of several benchmarks (e.g., [1, 2, 3, 4, 5, 6, 7]) that can assess the performance of SPARQL query processing systems. However, all of these benchmarks (except FedBench [5]) focus on the problem of query evaluation over local, centralised repositories. Hence, these benchmarks cannot be considered for benchmarking federated queries over multiple interlinked datasets hosted by different SPARQL endpoints. Moreover, many (e.g., [1, 2, 3, 6]) of them either rely on synthetic data or synthetic queries.

While synthetic benchmarks allow the generation of datasets of virtually any size to test the scalability of the systems, several works [8, 9, 10, 11, 12] point out that synthetic queries most commonly fail to reflect the characteristics of the real queries. Moreover, artificial benchmarks are typically highly structured while real Linked Data sources are less structured [8]. Hence, these synthetic benchmarks are not suited for evaluating how systems perform under realistic loads. A trend towards benchmarks with real data and real queries (e.g., FedBench [5], DBPSB [4], BioBenchmark [7]) has thus been pursued over recent years but has so far not been able to produce federated SPARQL query benchmarks that reflect the data volumes and query complexity that federated query engines already have to deal with on the Web of Data. In addition, most of the current benchmarks for SPARQL query execution focus on a single performance criterion, i.e., the query execution time. Thus, they fail to provide results that

*Corresponding author
Email addresses: saleem@informatik.uni-leipzig.de (Muhammad Saleem), ali.hasnain@insight-centre.org (Ali Hasnain), axel.ngonga@upb.de (Axel-Cyrille Ngonga Ngomo)

allow a fine-grained evaluation of SPARQL query processing systems to detect the components of systems that need to be improved [13, 14, 15]. For example, performance metrics such as

1. the *completeness and correctness of result sets*,
2. the *effectiveness of source selection* both in terms of *total number of data sources selected*, and
3. the corresponding *source selection time* (which both have a direct impact on the overall query performance)

are not addressed in the existing federated SPARQL query benchmarks [13, 14, 15].

In this paper, we present LargeRDFBench, a SPARQL endpoint federation benchmark that comprises real interlinked datasets from multiple domains. Our benchmark provides multiple performance measures as well as a set of queries with varying complexities. With this benchmark, we aim to provide means to test the different components of federation engines within an evaluation environment that closely reflects reality. Overall, our contributions are as follow:

- LargeRDFBench is an open-source benchmark for SPARQL endpoint query federation. To the best of our knowledge, this is the first federated SPARQL query benchmark with real data (from multiple interlinked datasets pertaining to different domains) to encompass more than 1 billion triples.
- We provide three types of queries, namely simple (from FedBench), complex, and large queries. These queries allow the evaluation of different aspects of the scalability of the current query federation frameworks. All queries are provided in SPARQL 1.0 and SPARQL 1.1 versions. Both versions represent exactly the same query and lead to exactly the same result set. The only difference lies in the SPARQL 1.1 version containing explicit `SERVICE` clauses, thus making a source selection unnecessary. Therewith, our benchmark allows for the comparison of federation engines along the axes of query execution time with and without source selection.
- We evaluate state-of-the-art SPARQL endpoint federation systems by using LargeRDFBench against several metrics including the source selection time, number of sources selected, result set correctness and completeness, the number of endpoint requests, and the query runtime. This fine-grained evaluation allows us to pinpoint the restrictions of current SPARQL endpoint federation systems when faced with large datasets, large intermediate results and large result sets.
- We show that the ranking of these systems based on benchmarks (i.e., FedBench) with simple queries differs significantly from their ranking on more complex queries. Moreover, our results also suggest that current state-of-the-art federation engines are not up to the challenge of dealing with large data queries, i.e., with queries that involve processing large intermediate result sets or lead to large result sets.

The rest of this paper is structured as follows: We begin by providing an overview of the main components of a SPARQL query federation benchmark (short: benchmark) and key features that need to be considered while designing such a benchmark. Then, we point out the current drawbacks of existing benchmarks in more detail (Section 3). In Section 4, we describe LargeRDFBench. In particular, we present the datasets and queries contained in the benchmark as well as the metrics used for benchmarking with LargeRDFBench. An evaluation of state-of-the-art systems based on LargeRDFBench follows next. The results are discussed before we make our conclusions. The benchmark and complete evaluation results can be found at `https://github.com/AKSW/largerdfbench`.

2. Background

This section presents the main components of SPARQL query processing benchmarks and the key features of each of these components that should be considered during the benchmark creation. In general, a SPARQL query benchmark can be regarded as consisting of three main components:

1. a set of RDF datasets,
2. a set of SPARQL queries and
3. a set of performance metrics.

Datasets: A federated benchmark should comprise of more than one dataset since a federated query is one that collects results from more than one dataset. Additionally, the datasets should vary in terms of the total number of triples, number of classes, number of resources, number of properties,

number of objects, average properties and instances per class, average indegrees and outdegrees as well as their distribution across resources [8]. Duan et al. [8] combines many of these datasets' features into a single composite metric called *structuredness* or *coherence*. For a given dataset, the structuredness value lies in [0,1], where 0 stands for the smallest possible amount of structure and 1 points to a perfectly structured dataset. A federated SPARQL query benchmark should comprise datasets of varying structuredness values.

SPARQL Queries: To present the key SPARQL query features that should be considered while designing a SPARQL querying benchmark, we begin by representing each basic graph pattern (BGP) of a SPARQL query as a directed hypergraph (DH) according to [15]. We chose this representation because it allows representing property-property joins (i.e., joins between the predicates of two or more triple patterns of a SPARQL query), which representations used in previous works [1, 16] do not allow to model. A BGP is formally defined as follows:

Definition 2.1 (Basic Graph Pattern)**.** We use the term basic graph pattern (BGP) exactly as per the SPARQL specification.[1] Formally, a BGP is defined as a set of triple patterns, where a triple pattern is defined as follows: Assume there are infinite and pairwise disjoint sets I (set of IRIs), B (set of blank nodes), L (set of literals) and V (set of variables). Then, a tuple from $(I \cup V \cup B) \times (I \cup V) \times (I \cup L \cup V \cup B)$ is a *triple pattern*. A sequence of triple patterns with optional filters is considered a single BGP. As per the specification of BGPs, any other graph pattern (e.g., `UNION`, `MINUS`, etc.) terminates a basic graph pattern.

The DH representation of a BGP is formally defined as follows:

Definition 2.2. Each basic graph patterns BGP_i of a SPARQL query can be represented as a DH $HG_i = (V, E, \lambda_{vt})$, where

- $V = V_s \cup V_p \cup V_o$ is the set of vertices of HG_i, V_s is the set of all subjects in HG_i, V_p the set of all predicates in HG_i and V_o the set of all objects in HG_i;
- $E = \{e_1, \ldots, e_t\} \subseteq V^3$ is a set of directed hyperedges (short: edge). Each edge $e = (v_s, v_p, v_o)$ emanates from the triple pattern $<v_s, v_p, v_o>$ in BGP_i. We represent these edges by connecting the head vertex v_s with the tail hypervertex (v_p, v_o). We use $E_{in}(v) \subseteq E$ and $E_{out}(v) \subseteq E$ to denote the set of incoming and outgoing edges of a vertex v. Formally, $E_{in}(v)$ contains exactly all edges $(v_s, v_p, v_o) \in E$ such that $v = v_p$ or $v = v_o$. $E_{out}(v)$ contains exactly all edges $(v_s, v_p, v_o) \in E$ such that $v = v_s$.
- λ_{vt} is a vertex-type-assignment function. A vertex $v \in V$ can be of type 'star', 'path', 'hybrid', or 'sink' if this vertex participates in at least one join. A 'star' vertex has more than one outgoing edge and no incoming edge. A 'path' vertex has exactly one incoming and one outgoing edge. A 'hybrid' vertex has either more than one incoming and at least one outgoing edge or more than one outgoing and at least one incoming edge. A 'sink' vertex has more than one incoming edge and no outgoing edge. A vertex that does not participate in any join is of type 'simple'.

The representation of a complete SPARQL query as DH is the union of the representations of its BGPs. As an example, a query and its DH representation are shown in Figure 1. Based on the DH representation of SPARQL queries, we can define:

Theorem 1 (Number of Triple Patterns)**.** From Definition 2.2, the total number of triple patterns in a BGP_i is equal to the number of hyperedges $|E|$ in the DH representation of the BGP_i.

Proof of Theorem 1. Each hyperedge connects the head vertex (which represents the subject of the triple pattern) and tail hyper vertex (which represents the predicate and object of the same triple pattern). Thus, for each triple pattern, a hyperedge is created in the DH representation of the query. Consequently, the number of hyperedges in the DH representation of BGP is equal to the number of triple patterns in that BGP. □

The total number of triple patterns in a query is the sum of the total number of triple patterns across all of the BGPs contained in this query.

Definition 2.3 (Number of Join Vertices)**.** Let $ST = \{st_1, \ldots, st_j\}$ be the set of vertices of type 'star', $PT = \{pt_1, \ldots, pt_k\}$ be the set of vertices of type 'path', $HB = \{hb_1, \ldots, hb_l\}$ be the set of vertices of type 'hybrid', and $SN = \{sn_1, \ldots, sn_m\}$ be the

[1] See `https://www.w3.org/TR/sparql11-query/#BasicGraphPatterns`

```
SELECT DISTINCT *
WHERE
{
?drug :description ?drugDesc.
?drug :drugType :smallMolecule.
?drug :keggCompoundId ?compound.
?enzyme :xSubstrate ?compound.
?Chemicalreaction :xEnzyme ?enzyme.
?Chemicalreaction :equation
                ?ChemicalEquation.
?Chemicalreaction :title
                  ?ReactionTitle
 }
```

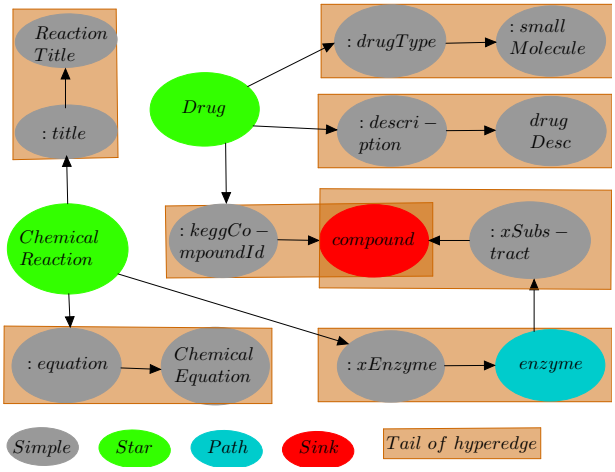


Figure 1: DH representation of the SPARQL query. Prefixes are ignored for simplicity

set of vertices of type 'sink' in a DH representation of a SPARQL query, then the total number of join vertices in the query $\#JV = |ST| + |PT| + |HB| + |SN|$.

The total number of join vertices in a query is the sum of the total number of join vertices across all of the BGPs contained in this query.

Definition 2.4 (Join Vertex Degree)**.** The DH representation of SPARQL queries makes use of the notion of $E_{in}(v) \subseteq E$ and $E_{out}(v) \subseteq E$ to denote the set of incoming and outgoing hyperedges of a vertex v. The join vertex degree of a vertex v is denoted $JVD_v = |E_{in}(v)| + |E_{out}(v)|$.

The join vertex degree of the complete query is the average of join vertex degree of all the joins contained in this query. In our example (see Figure 1), the number of triple patterns is seven and the number of join vertices is four (two star, one sink and path each). The join vertex degree of each of the 'star' join vertex (shown in green color) given in Figure 1 is three (i.e., three outgoing hyperedges from both vertices).

Definition 2.5 (Relevant Source Set)**.** Let D be the set of all data sources (e.g., SPARQL endpoints), TP be the set of all triple patterns in query Q. Then, a source $d \in D$, is *relevant* (also called *capable*) for a triple pattern $tp_i \in TP$ if at least one triple contained in d matches tp_i.[2] The relevant source set $R_i \subseteq D$ for tp_i is the set that contains all sources that are relevant for that particular triple pattern.

Definition 2.6 (Total Triple Pattern-wise Sources)**.** By using Definition 2.5, we can define the total number of triple pattern-wise sources selected for query Q as the sum of the magnitudes of relevant source sets R_i over all individual triple patterns $tp_i \in$ Q.

Definition 2.7 (Number of Relevant Sources)**.** The sources that potentially contribute to the result set of a query are those that are *relevant* to at least one triple pattern in the query [17], i.e. a source is relevant to the query if it is able to provide at least one result for any of the triple patterns of the query.

For the query S2 (3 triple patterns, ref. page 27) given in the appendix, the *relevant source set* for the first and third triple patterns only contains $DBpedia{-}Subset$ and the *relevant source set* for the second triple pattern only contains $NewYorkTimes$. The *total triple pattern-wise sources selected* for this query is equal to 3, i.e., the sum of relevant sources for individual triple patterns. The distinct *number of relevant sources* for this query is two, i.e., $DBpedia - Subset$ and $NewYorkTimes$.

Definition 2.8 (Triple Pattern Selectivity)**.** Let tp_i be a triple pattern and d be a *relevant* source for tp_i. Furthermore, let N be the total number of triples in d and $Card(tp_i, d)$ be the cardinality of tp_i w.r.t. d, i.e., total number of triples in d that matches tp_i, then the selectivity of tp_i w.r.t. d denoted by $Sel(tp_i, d) = Card(tp_i, d)/N$. The selectivity of the

[2]The concept of matching a triple pattern is defined formally in the SPARQL specification found at `http://www.w3.org/TR/rdf-sparql-query/`

filtered triple pattern (a triple pattern with SPARQL `FILTER` clause) is calculated in the same way.

Definition 2.9 (BGP-Restricted Triple Pattern Selectivity)**.** Consider a Basic Graph Pattern BGP and a triple pattern tp_i belonging to BGP, let $R(tp_i, d)$ be the set of distinct solution mappings (i.e., resultset) of executing tp_i over dataset d and $R(BGP, d)$ be the set of distinct solution mappings of executing BGP over dataset d. Then the BGP-restricted triple pattern selectivity denoted by $Sel_{BGP-Restricted}(tp_i, d)$, is the fraction of distinct solution mappings in $R(tp_i, d)$ that are compatible (as per standard SPARQL semantics [18]) with a solution mapping in $R(BGP, d)$ [1]. Formally, if Ω and Ω' denote the sets underlying the (bag) query results $R(tp_i, d)$ and $R(BGP, d)$, respectively, then

$$Sel_{BGP-Restricted}(tp_i, d) = \frac{|\{\mu \in \Omega \mid \exists \mu' \in \Omega' : \mu \text{ and } \mu' \text{ are compatible}\}|}{|\Omega|}$$

Definition 2.10 (Join-Restricted Triple Pattern Selectivity)**.** Consider a join vertex x in the DH representation of a Basic Graph Pattern BGP. Let BGP' belonging to BGP be the set of triple patterns that are incidents to x. Furthermore, let tp_i belonging to BGP' be a triple pattern and $R(tp_i, d)$ be the set of distinct solution mappings of executing tp_i over dataset d and $R(BGP', d)$ be the set of distinct solution mappings of executing BGP' over dataset d. Then the $x - restricted$ triple pattern selectivity denoted by $Sel_{JVx-Restricted}(tp_i, d)$, is the fraction of distinct solution mappings in $R(tp_i, d)$ that are compatible with a solution mapping in $R(BGP', d)$ [1]. Formally, if Ω and Ω' denote the sets underlying the (bag) query results $R(tp_i, d)$ and $R(BGP', d)$, respectively, then

$$Sel_{JVx-Restricted}(tp_i, d) = \frac{|\{\mu \in \Omega \mid \exists \mu' \in \Omega' : \mu \text{ and } \mu' \text{ are compatible}\}|}{|\Omega|}$$

According to previous works [1, 16], a federated SPARQL query benchmark should vary the queries it contains w.r.t. the following *query characteristics*: number of triple patterns, number of join vertices, mean join vertex degree, number of sources span, query result set sizes, mean triple pattern selectivities (should be mean Filtered triple pattern selectivities if SPARQL `FILTER` clause is attached to the triple pattern), BGP-restricted triple pattern selectivity, join-restricted triple pattern selectivity, join vertex types ('star', 'path', 'hybrid', 'sink'), and SPARQL clauses used (e.g., `LIMIT`, `OPTIONAL`, `ORDER BY`, `DISTINCT`, `UNION`, `FILTER`, `REGEX`).

Performance Metrics: Previous works [14, 15] show that the *result set completeness and correctness*, the *total triple pattern-wise sources selected*, the *number of SPARQL ASK requests* used during source selection, the *source selection time*, the number of endpoint requests, and the *overall query execution time* are important metrics to be considered in SPARQL query federation benchmarks. These metrics are general and applicable to any SPARQL endpoint federation engine. We thus decided to implement these measures in LargeRDFBench. We did not consider the number of endpoint requests as they strongly depend upon the configuration of the engine, especially the block size (e.g., 15 is used in FedX by default, doubling the block size will reduce the number of endpoint request by 50%) and buffer size used. Furthermore, the network latency is important to consider for live SPARQL endpoints query processing. However, it is almost negligible for local, dedicated network setup used in our evaluation. A utility that calculates all of the above benchmark's key features is provided at the project home page along with usage instructions.

Finally, it is important to make a distinction between the two well-known categories of SPARQL query federation: *SPARQL endpoint federation* and *Linked Data federation* [15]. In the former type of federation, the RDF data is made available via SPARQL endpoints, the sub-queries are directly forwarded to the SPARQL endpoints, and the corresponding results are integrated by using different join techniques. The advantage of this category of approaches is that the execution of queries can be carried out efficiently because the approach relies on SPARQL endpoints. Furthermore, the queries are answered based on original, up-to-date data with no synchronization of the copied data required [19]. However, for such approaches to work, the data needs to be made available through SPARQL endpoints. Thus, SPARQL query federation approaches are unable to deal with the data provided by the whole of the LOD Cloud because data is partly not exposed through SPARQL endpoints. In Linked Data federation approaches, the data does not need to be exposed via SPARQL endpoints. The only requirement is that the data should follow the Linked

Data principles[3]. However, due to URI's lookups at runtime, these type of approaches are usually slower than SPARQL query federation approaches. In this work, we are interested in designing a benchmark for *SPARQL endpoint federation* approaches.

3. Related work

Benchmarks for measuring the advance of SPARQL query processing engines has been regarded as central for the development of the Semantic Web since its creation. Consequently, a good number of benchmarks for comparing SPARQL query processing systems have been developed over the last decade. These include the Waterloo Stress Testing Benchmark (WSTB) [1], the Berlin SPARQL Benchmark (BSBM) [2], the Lehigh University Benchmark (LUBM) [3], the DBpedia Sparql Benchmark (DBPSB) [4], FedBench [5], SP^{2}Bench [6], and the BioBenchmark [7]. In addition, FEASIBLE [9] is a customizable SPARQL benchmark generation framework out of SPARQL queries log. WSTB, BSBM, DBPSB, SP^{2}Bench, FEASIBLE, and BioBenchmark were designed with the main goal of evaluating query engines that access data kept in a single repository. They are used for the performance evaluation of different triple stores. LUBM was designed for comparing the performance of OWL reasoning engines. However, all of these benchmarks do not consider distributed data and federated SPARQL queries, thus they are not further considered in the discussion.

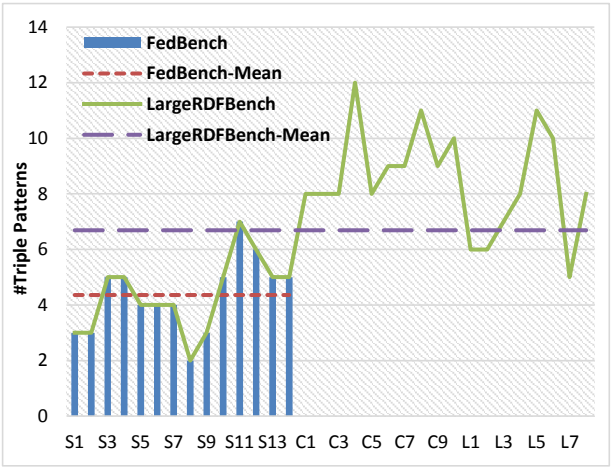
SPLODGE [16] is a heuristic for the automatic generation of federated queries with conjunctive BGPs. Non-conjunctive queries that make use of the SPARQL `UNION`, `OPTIONAL` clauses are not considered. However, non-conjunctive queries are widely used in practice. For example, the DBpedia query log [11] contains 20.87% queries of SPARQL `UNION` and 30.02% queries of SPARQL `FILTER` clauses. Moreover, the use of different SPARQL clauses and triple pattern join types greatly varies from one dataset to another dataset, thus making it very difficult for an automatic query generator to reflect the reality. For example, the DBpedia and Semantic Web Dog Food (SWDF) query log [10] differ significantly in their use of `LIMIT` (27.99% for SWDF vs 1.04% for DBpedia) and `OPTIONAL` (0.41% for SWDF vs 16.61% for DBpedia) clauses.

To the best of our knowledge, FedBench is the only benchmark that encompasses real-world datasets, commonly used federated SPARQL queries and a distributed data environment. It comprises a total of 14 queries for SPARQL endpoint federation and 11 queries for Linked Data federation approaches. In addition, this benchmark includes a dataset and queries from SP^{2}Bench. FedBench is commonly used in the evaluation of SPARQL query federation systems [17, 20, 21, 15, 22, 23, 24, 25]. However, the real queries (excluding synthetic SP^{2}Bench benchmark queries) are low in complexity (in terms of total triple patterns, query result set sizes, execution time, use of SPARQL features etc.). The 11 Linked Data federation queries do not make use of any of the SPARQL clauses given in Table 1, the number of triple patterns included in the query ranges from 2 to 5, and the query result set sizes only ranges from 1 to 1216 (6/11 queries having result set size less than 51). As mentioned before, we are only interested in the 14 SPARQL endpoint federation queries. Hence, only those are further discussed in rest of the paper.

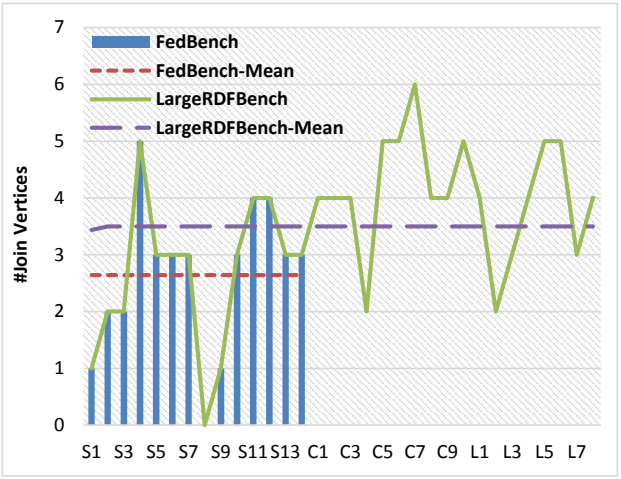
Table 1, Figure 2, and Figure 3d show that the FedBench SPARQL endpoint federation queries are also low in complexity and do not sufficiently complement each other (the systems should not be tested with queries identical in properties). Consequently, they favour (as shown by our evaluation in Section 5.2.4) particular types of federation systems.

The number of Triple Patterns (#TP, ref. Figure 2a) included in the query ranges from 2 to 7. Consequently, the standard deviations of the number of Join Vertices (#JV, ref. Figure 2b), the Mean Join Vertices Degrees (#MJVD, ref. Figure 2c), and the number of Sources the query Span (#SS, ref. Figure 2d) are low. In particular, there are: 6/14 queries with #JV exactly equal to 3, 8/14 queries with #MJVD exactly equal to 2, and 5/14 queries with #SS exactly equal to 2. The query result set sizes (#R, ref. Figure 3a) are small (maximum 9054, 6/14 queries lead to a result set whose magnitude is less than 4). The query triple patterns are not highly selective in general (ref. Figure 3b). The important SPARQL clauses such as `DISTINCT`, `ORDER BY` and `REGEX` are not used (ref. Table 1). Moreover, the SPARQL `OPTIONAL` and `FILTER` clauses are only used in a single query (i.e., LS7 of FedBench). FedBench shows good variation in mean (across all triple patterns of the query) BGP-restricted (ref. Figure 3c) and mean join-restricted (ref. Figure 3d) triple pattern selectivities. However, the mean join-

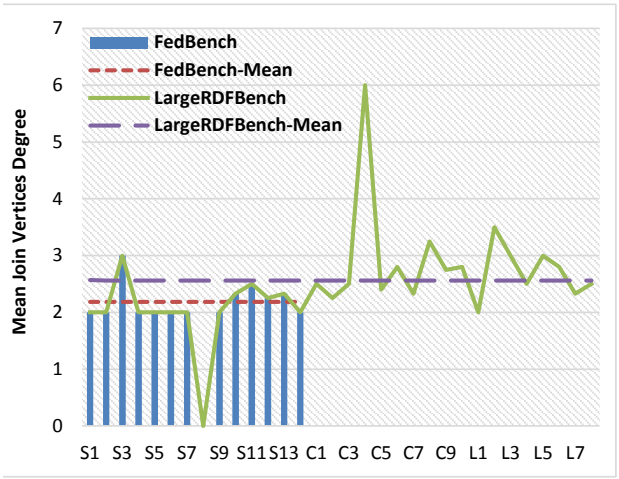
[3] `http://www.w3.org/DesignIssues/LinkedData.html`



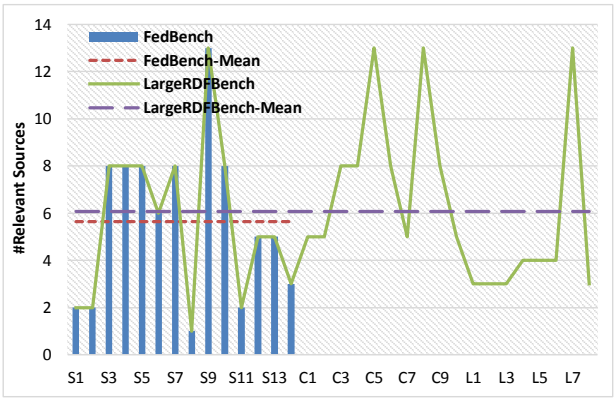
(a) #TP. (±1.33, ±2.64)



(b) #JV. (±1.33, ±1.41)

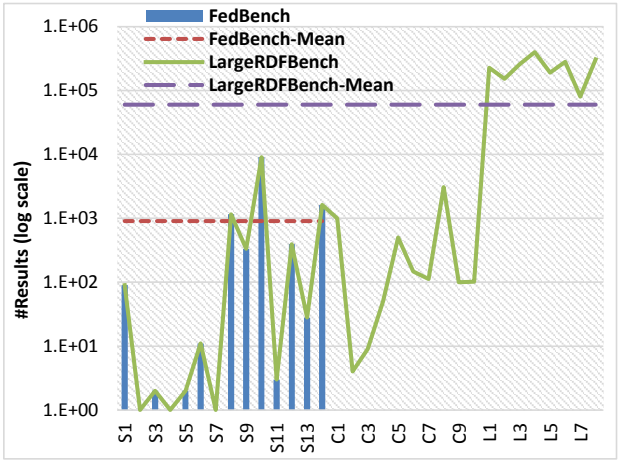


(c) MJVD. (±0.30, ±0.76)

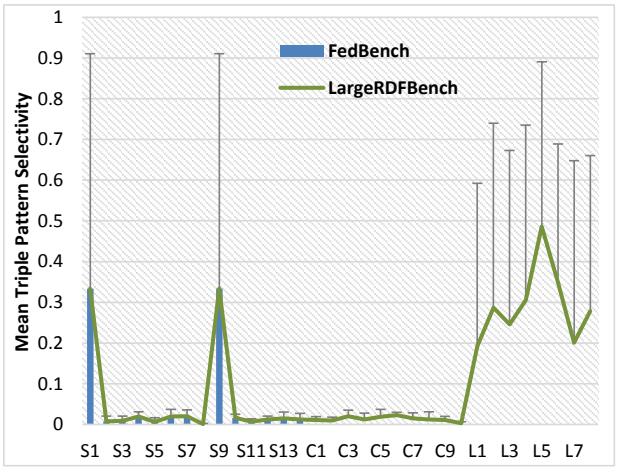


(d) #RS. (±3.41, ±3.44)

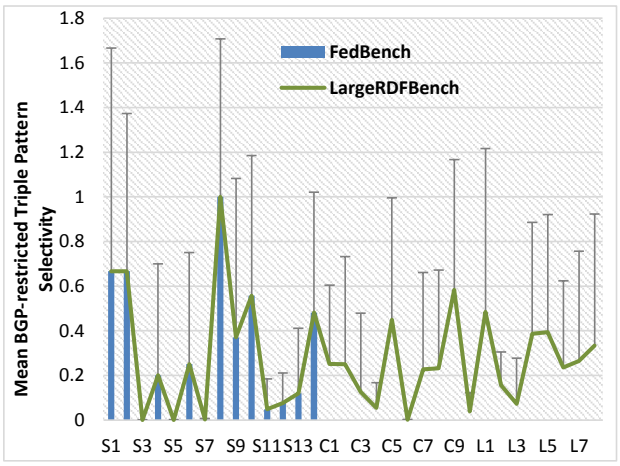
Figure 2: Comparison of structural query characteristics of FedBench and LargeRDFBench. **#TP** = Number of triple patterns, **#JV** = Number of join vertices, **MJVD** = Mean join vertices degree, **#RS** = Number of relevant sources. The values inside bracket show standard deviations for FedBench and LargeRDFBench, respectively. X-axis shows the query name.



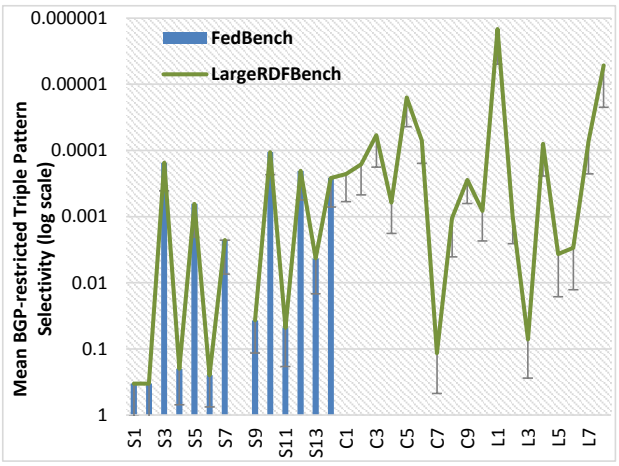
(a) #R. (±2397, ±113763)



(b) MTPS. (±0.11, ±0.14)



(c) MBRTPS. (±0.31, ±0.24)



(d) MJRTPS. (±0.13, ±0.09)

Figure 3: Comparison of data-driven characteristics of FedBench and LargeRDFBench. **#R** = Number of results, **MTPS** = Mean Triple Pattern Selectivity, **MTPS** = Mean Triple Pattern Selectivity, **MBRTPS** = Mean BGP-restricted Triple Pattern Selectivity, **MJRTPS** = Mean Join-restricted Triple Pattern Selectivity. The values inside bracket shows standard deviations for FedBench and LargeRDFBench, respectively. X-axis shows the query name.

Table 1: Queries distribution with respect to SPARQL clauses and join vertex types.

	SPARQL Clauses							Join Vertex Type			
Benchmark	LIMIT	OPTIONAL	ORDER BY	DISTINCT	UNION	FILTER	REGEX	Star	Path	Hybrid	Sink
FedBench	0%	7.14%	0%	0%	21.42%	7.14%	0%	85.71%	57.14%	14.28%	35.71%
LargeRDFBench	12.5%	25%	9.3%	28.1%	18.75%	31.25%	3.12%	75%	78.12%	40.62%	40.62%

restricted selectivities are in general high, meaning most of the solution mappings of the triple pattern are compatible with the solution mappings of other triple patterns incident on the same join variable. This is because the mean join vertex degree (ref. Figure 2c) of the FedBench queries is in general lower in comparison to LargRDFBench queries. Most importantly, the average query execution is small (about 2 seconds on average ref. Section 5.2.4). Finally, FedBench relies only on the number of endpoints requests and the query execution time as performance criteria. These limitations make it difficult to extrapolate how SPARQL query federation engines will perform when faced with the growing amount of data available on the Data Web based on FedBench results. A fine-grained evaluation of the federation engines to detect the components that need to be improved is also not possible [14].

Our benchmark includes all of the 14 SPARQL endpoint federation queries (which we named *simple queries*) from FedBench, as they are useful but not sufficient all alone. In addition, we provide 10 complex and 8 large data queries, which lead to larger result sets (see Figure 3a) and intermediary results (see triple pattern selectivities, Figure 3b). Beside the central performance criterion, i.e., the query execution time, our benchmark includes result set completeness and correctness, effective source selection in terms of the total number of data sources selected, the total number of SPARQL ASK requests used and the corresponding source selection time. Our evaluation results (section 5.2) suggest that the performance of current SPARQL query federation systems on simple queries (i.e., FedBench queries) does not reflect the systems' performance on more complex queries. In addition, none of the state-of-the-art SPARQL query federation systems is able to fully answer the real use-case large data queries.

4. Benchmark Description

To address the aforementioned limitations, we propose LargeRDFBench, a billion-triple benchmark which encompasses a total of 13 real, interconnected datasets of varying *structuredness* (ref. Figure 5) and 32 real queries of varying complexities (see Table 1 and Figure 2) ranging from simple to complex. The idea behind this work was to design a benchmark based on real data and real queries that implements all of the key benchmark features discussed in Section 2. The data was chosen to reflect the topology of the current Web of Data, with some of the datasets being highly connected with other datasets while others are isolated (ref. Figure 4). Furthermore, some of the datasets are highly *structured* while others are low *structured* (ref. Figure 5). The queries were chosen to reflect a wide range of complexities w.r.t. the number of triple patterns they contain, the use of different SPARQL clauses, the triple patterns' selectivity, the number of join vertices, the mean join vertices degrees, the number of sources span, and the result set sizes they lead to (see Table 1 and Figure 2). The resulting benchmark, dubbed LargeRDFBench, consists consequently of three main components: (1) real-world datasets collected from different domains, (2) queries showing typical requests, mostly collected from domain experts and/or representing real use cases, and (3) a comprehensive set of fine-grained evaluation measures. In the following section, we present each of the three main components in detail.

4.1. Benchmark Datasets

Our benchmark consists of a total of 13 real-world datasets[4] of which 12 are interlinked. The datasets were collected from different domains as shown in Figure 4. We began by selecting all nine real-world datasets from Fedbench [5]. We added three subdatasets from three different Linked TCGA (Cancer Genome Atlas) live SPARQL endpoints[5] [26] (i.e. Linked TCGA-A, Linked TCGA-M, and Linked TCGA-E) along with Affymetrix[6]. We chose Linked TCGA because it is one of the first datasets that abides by many of the Vs of large data (Volume,

[4]Live SPARQL endpoints, datadumps URL and more details about the benchmark at homepage

[5]`http://tcga.deri.ie/`

[6]`http://download.bio2rdf.org/release/2/affymetrix/affymetrix.html`

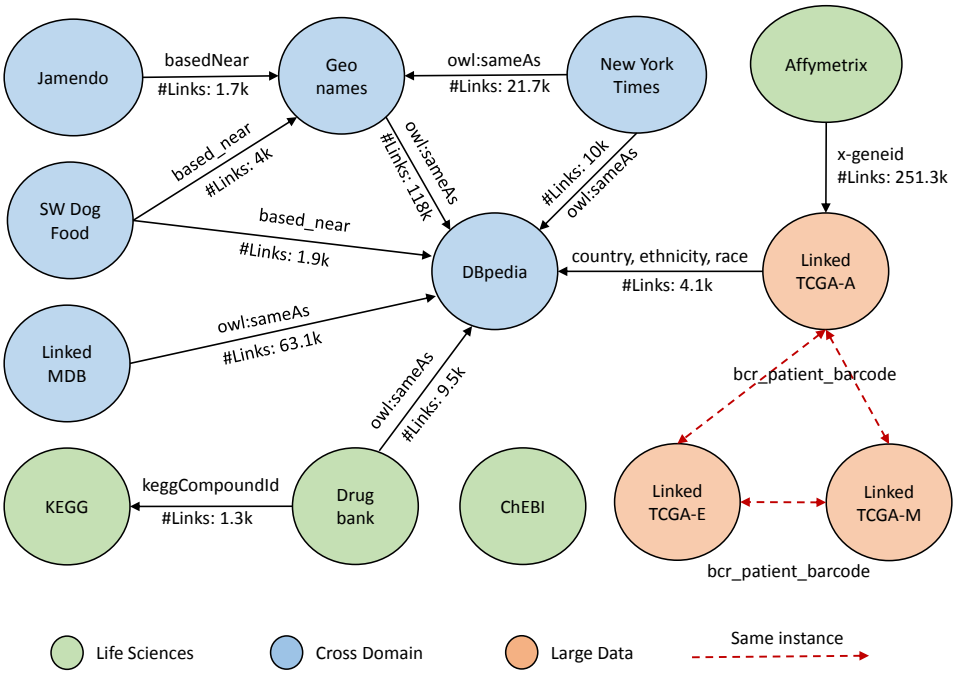


Figure 4: LargeRDFBench datasets connectivity

Veracity, Value, ...) [27, 26, 28]. Moreover, Linked TCGA has a large number of links to Affymetrix, which we thus added to the list of our datasets. The addition of these four datasets enabled us to include real federated queries with large result set sizes (minimum 80459, see Figure 3a) into the benchmark.

Figure 4 shows the topology of all 13 datasets in LargeRDFBench. Other basic statistics like the total number of triples, the number of resources, predicates and objects, as well as the number of classes and the number of links can be found in Table 2. Note that ChEBI has no link with any other benchmark dataset. However, its predicate "title" and DrugBank's predicate "genericName" display the same literal values. Similarly, the Linked TCGA-A predicate "drug_name" and DrugBank's "genericName" display the same values. Thus, they can be used in federated SPARQL queries. Furthermore, each Linked TCGA patient (uniquely identified by bcr_patient_barcode) expression data is distributed across the three large data datasets, i.e., Linked TCGA-A, Linked TCGA-E, and Linked TCGA-M (further explained in 4.1.3 subsection Large Data). The datasets in LargeRDFBench belong to three categories: Cross-domain, Life Sciences domain and large data.

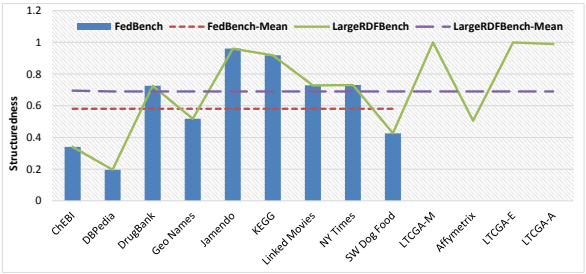


Figure 5: LargeRDFBench datasets structuredness (±0.26, ±0.27)

4.1.1. Cross-domain Datasets

This category comprises datasets which pertain to several domains including news, movies, music, semantic web conferences and geography. These datasets include 1) a subset from DBpedia[7] comprising infoboxes and the instance types, 2) the music knowledge base called Jamendo[8], 3) LinkedMDB[9] , a knowledge base containing movie and actor information, 4) GeoNames[10] , which contains geographical data about persons, locations, as well as organisa-

[7] http://DBpedia.org/
[8] http://dbtune.org/jamendo/
[9] http://linkedmdb.org/
[10] http://www.geonames.org/

Table 2: LargeRDFBench datasets statistics. Structuredness is calculated according to [8] and is averaged in the last row.

Dataset	#Triples	#Subjects	#Predicates	#Objects	#Classes	#Links	Structuredness
Linked TCGA-M	415,030,327	83,006,609	6	166,106,744	1	-	1
Linked TCGA-E	344,576,146	57,429,904	7	84,403,422	1	-	1
Linked TCGA-A	35,329,868	5,782,962	383	8,329,393	23	251.3k	0.99
ChEBI	4,772,706	50,477	28	772,138	1	-	0.340
DBpedia-Subset	42,849,609	9,495,865	1,063	13,620,028	248	65.8k	0.196
DrugBank	517,023	19,693	119	276,142	8	9.5k	0.726
Geo Names	107,950,085	7,479,714	26	35,799,392	1	118k	0.518
Jamendo	1,049,647	335,925	26	440,686	11	1.7k	0.961
KEGG	1,090,830	34,260	21	939,258	4	30k	0.919
LinkedMDB	6,147,996	694,400	222	2,052,959	53	63.1k	0.729
New York Times	335,198	21,666	36	191,538	2	31.7k	0.731
Semantic Web Dog Food	103,595	11,974	118	37,547	103	1.6k	0.426
Affymetrix	44,207,146	1,421,763	105	13,240,270	3	246.3k	0.506
Total/Average	**1,003,960,176**	**165,785,212**	**2,160**	**326,209,517**	**459**	**818.7k**	**0.65**

tions, 5) Semantic Web Dog Food[11] which describes Semantic Web conferences and publications, and 6) a knowledge base containing news from the New York Times[12]. Figure 4 shows the links between these data sources.

4.1.2. Life Sciences Domain

The life sciences domain contains datasets pertaining to human beings, healthcare and life sciences. Different data sources were selected from the life sciences domain dealing with chemical compounds, genomes, gene expressions, reactions and drugs. Life Sciences domain data sources include:

1) Drugbank[13], which is a knowledge base containing information pertaining to drugs, their composition and their interactions with other drugs. Drugbank is a resource for bioinformatics and cheminformatics containing details regarding drugs, i.e., chemical, pharmacological and pharmaceutical. It also contains data with drug target information i.e. sequence, structure, and pathway. At present the original drugbank database contains 8261 drug entries including Food and Drug Administration (FDA) approved small molecule drugs, FDA approved biotech (protein/peptide) drugs, nutraceuticals and over 6000 experimental drugs. It also contains 4338 non-redundant protein containing target, enzyme, transporter and carrier sequences.

2) Kyoto Encyclopedia of Genes and Genomes (KEGG)[14] which contains further information about chemical compounds and reactions with a focus on information relevant for geneticists. KEGG database provides resources for understanding abstract level functions of any biological system. It includes the cell, the organism and the ecosystem. The information is collected from the molecular-level, especially large-scale molecular datasets generated by genome sequencing and other experimental technologies.

3) Chemical Entities of Biological Interest (ChEBI)[15] knowledge base which describes the life sciences domain from a chemical point of view. ChEBI focuses on small chemical compounds including isotopically distinct atoms, molecules, ions, ions pairs, radicals, radical ions, complex, conformers, etc. These can either be collected from natural sources or through synthetic products.

4) Affymetrix[16] dataset that contains the probesets used in the Affymetrix microarrays.Affymetrix microarrays provides high density oligo-nucleotide gene expression arrays. Each gene on an Affymetrix microarray is typically represented by a probeset consisting of 11 different pairs of 25 oligo-nucleotide that represents the features of the transcribed region of particular gene under consideration [29].

4.1.3. Large Data: Linked TCGA

Linked TCGA is the RDF version of Cancer Genome Atlas[17] (TCGA) presented in [26]. This knowledge base contains cancer patient data generated by the TCGA pilot project, started in 2005 by the National Cancer Institute (NCI) and the National Human Genome Research Institute (NHGRI).

[11] `http://data.semanticweb.org/`
[12] `http://www.nytimes.com/`
[13] `http://www.drugbank.ca/`
[14] `http://www.genome.jp/kegg/`
[15] `https://www.ebi.ac.uk/chebi/`
[16] `http://www.affymetrix.com/`
[17] `http://cancergenome.nih.gov/`

Currently, Linked TCGA comprises a total of 20.4 billion triples[18] from 9000 cancer patients and 27 different tumour types. For each cancer patient, Linked TCGA contains expression results for the DNA methylation, Expression Exon, Expression Gene, miRNA, Copy Number Variance, Expression Protein, SNP, and the corresponding clinical data.

Given that we aimed to build a 1-billion-triple dataset, we selected 306 patient data randomly to reach the targeted 1 billion triples. The patients distributed evenly across 3 different cancer types, i.e. Cervical (CESC), Lung squamous carcinoma (LUSC) and Cutaneous melanoma (SKCM). The selection of the patients was carried out by consulting domain experts. This data is hosted in three TCGA SPARQL endpoints with all DNA methylation data in the first endpoint, all Expression Exon data in the second endpoint, and the remaining data in the third endpoint. Consequently, we created three different datasets, namely the Linked TCGA-M, Linked TCGA-E, and Linked TCGA-A containing methylation, exon, and all remaining data, respectively. Further statistics about these three datasets can be found at the project homepage.

4.2. Benchmark Queries

LargeRDFBench comprises a total of 32 queries for *SPARQL endpoint federation approaches*. These queries are divided into three different types: the 14 simple queries (namely S1-S14) are from FedBench (CD1-CD7 and LS1-LS7). The 10 complex queries (namely C1-C10) and the 8 large data queries (dubbed L1-L8) were created by the authors with the help of domain experts. All of the queries are given in an Appendix at the end of the paper. Table 3 summarizes the results presented in Table 1, Figure 2, and Figure 3. In Table 3, we can see that the SPARQL 1.1 versions of the query sources (i.e., values inside the brackets of #RS) required to compute the complete result set of the query is smaller than the number of relevant sources. This is because it is possible that a source is relevant for a triple pattern of the query but its results may be excluded after performing the join with the results of another triple pattern in the same query [15]. As such this source only contributes to the intermediate results and does not contribute to the final result set of the query. However, the number of relevant sources is important since it has a direct impact on the number of intermediate results [15]. The minimum number of sources required to compute the complete result set of the query is explicitly annotated at the SPARQL 1.1 version of the queries given in the appendix. The number of relevant sources against the individual triple patterns of the query is given at the aforementioned LargeRDFBench homepage.

4.2.1. Simple Queries

In comparison to the other queries in the benchmark, the queries in this category comprise the smallest number of triple patterns, which range from 2 to 7 (average #TP = 4.3, ref. Table 3). These queries require the retrieval of data from 2 to 13 data sources. The number of join vertices and the mean join vertice degree for these queries are lower (average #JV = 2.6, MJVD = 2.1, ref. Table 3). Moreover, they only use a subset of the SPARQL clauses as shown in Table 1(see FedBench row as all of the simple queries are from FedBench). Amongst others, they do not use `LIMIT`, `REGEX`, `DISTINCT` and `ORDER BY` clauses. Finally, we will see in the evaluation section that the query execution time for such queries is small (around 2 seconds for FedX).

It is important to mention that we removed the `FILTER (?mass > '5')` from the FedBench life sciences query LS7 (S14 in LargeRDFBench) because the KEGG drug mass is a string. Thus, using this operator on KEGG would lead to semantics, different from that intended in the original query. Consequently, the result set size changes from 114 to 1620 rows.

4.2.2. Complex Queries

The complex queries were defined to address the restrictions of simple queries with respect to the number of triple patterns they use, the number of join vertices, the mean join vertices degree, the SPARQL clauses, and the small query execution times. Consequently, queries in this category rely on at least 8 triple patterns. The number of join vertices ranges from 3 to 6 (average #JV = 4.3, ref. Table 3). The mean join vertices degree ranges from 2 to 6 (average MJVD = 2.93, ref. Table 3). In addition, they were designed to use more SPARQL clauses, especially, `DISTINCT`, `LIMIT`, `FILTER` and `ORDER BY`. Later, we will see in the evaluation that the query execution time for complex queries exceeds 10 minutes.

[18] `http://tcga.deri.ie/`

Table 3: LargeRDFBench query characteristics. (**#TP** = total number of triple patterns in a query, **#RS** = distinct number of relevant source. The values inside brackets show the total number of distinct sources used in the SPARQL 1.1 version (using SPARQL SERVICE clause) of each of the benchmark queries. the minimum number of distinct sources required to get the complete result set, **#Results** = total number of results), **#JV** = total number join vertices, **MJVD** = mean join vertices degree, **MTPS** = mean triple pattern selectivity, **MBRTPS** = Mean BGP-restricted triple pattern selectivity, **MJRTPS** = mean join-restricted triple pattern selectivity, **UN** = UNION, **OP** = OPTIONAL, **DI** = DISTINCT, **FI** = FILTER, **LI** = LIMIT, **OB** = ORDER BY, **RE** = Regex, **NA** = not applicable since there is no join node in the query, **-** = no SPARQL clause used. **Avg.** = the average values across the individual queries categories, i.e., simple, complex, and large data.

Query	Join Vertices	#TP	#RS	#Results	#JV	MJVD	MTPS	MBRTPS	MJRTPS	Clauses
S1	1 Star	3	2(2)	90	1	2	0.333	0.66667	0.33333	UN
S2	1 Star,1 Path	3	2(2)	1	2	2	0.007	0.66671	0.33338	-
S3	1 Star,1 Hybrid	5	8(2)	2	2	3	0.008	0.00031	0.00015	-
S4	2 Star,2 Sink,1 Path	5	8(2)	1	5	2	0.019	0.20003	0.20003	-
S5	1 Star,2 Path	4	8(2)	2	3	2	0.006	0.00067	0.00064	-
S6	1 Star,2 Path	4	6(4)	11	3	2	0.019	0.25013	0.25001	-
S7	1 Star,2 Path	4	8(2)	1	3	2	0.020	0.00227	0.00227	-
S8	No Join	2	1(1)	1159	0	NA	0.001	1.00000	0.00000	UN
S9	1 Path	3	13(2)	333	1	2	0.333	0.37037	0.03704	UN
S10	1 Star,2 Path	5	8(2)	9054	3	2.33	0.016	0.55678	0.00011	-
S11	2 Star,1 Sink,1 Hybrid	7	2(2)	3	4	2.5	0.006	0.04800	0.04788	-
S12	2 Star,1 Path,1 Sink	6	5(3)	393	4	2.25	0.012	0.07633	0.00020	-
S13	3 Star	5	5(2)	28	3	2.33	0.014	0.11996	0.00428	-
S14	2 Star,1 Sink	5	3(2)	1620	3	2	0.012	0.48156	0.00026	OP
Avg.		4.3	5.7(2.1)	907	2.6	2.1	0.057	0.31713	0.08640	
C1	2 Star,1 Path,1 Sink	8	5(2)	1000	4	2.5	0.010	0.25162	0.00023	DI, FI, OP, LI
C2	2 Star,1 Path,1 Sink	8	5(3)	4	4	2.25	0.009	0.25065	0.00016	OP, FI
C3	2 Star,1 Path,1 Hybrid	8	8(3)	9	4	2.25	0.020	0.12542	0.00006	DI, OP
C4	2 Star	12	8(3)	50	2	6	0.0124	0.05407	0.00061	DI, OP, LI
C5	2 Star,2 Path,1 Sink	8	13(2)	500	5	2.4	0.0186	0.44883	0.00002	FI, LI
C6	2 Star,1 Path,2 Sink	9	8(2)	148	5	2.8	0.022	0.00103	0.00007	OB
C7	3 Star,1 Path,1 Sink,1 Hybrid	9	5(2)	112	6	2.33	0.014	0.22688	0.11615	DI, OP
C8	2 Star,1 Path,1 Hybrid	11	13(2)	3067	4	3.25	0.012	0.23173	0.00106	DI, OP
C9	2 Star,2 Path	9	8(2)	100	4	2.75	0.011	0.58352	0.00028	OP, OB, LI
C10	2 Star,2 Path,1 Hybrid	10	5(3)	102	5	2.8	0.002	0.03891	0.00082	DI
Avg.		9.2	7.8(2.4)	509.2	4.3	2.93	0.013	0.22127	0.01195	
L1	4 Path	6	3(2)	227192	4	2	0.192	0.48437	0.00001	UN
L2	1 Path,1 Hybrid	6	3(2)	152899	2	3.5	0.286	0.15652	0.00098	DI, FI
L3	2 Path,1 Hybrid	7	3(2)	257158	3	3	0.245	0.07255	0.07205	FI, OB
L4	2 Path,2 Hybrid	8	4(2)	397204	4	2.5	0.305	0.38605	0.00008	UN, FI, RE
L5	1 Star,1 Path,1 Sink,2 Hybrid	11	4(3)	190575	5	3	0.485	0.39364	0.00367	FI
L6	1 Star,1 Path,1 Sink,2 Hybrid	10	4(2)	282154	5	2.8	0.349	0.23553	0.00298	FI, DI
L7	2 Path,1 Hybrid	5	13(2)	80460	3	2.33	0.200	0.26498	0.00007	DI, FI
L8	2 Path,2 Hybrid	8	3(2)	306705	4	2.5	0.278	0.33376	0.00001	UN, FI
Avg.		7.62	4.6(2.1)	236793	3.75	2.70	0.293	0.29092	0.00998	

4.2.3. Large Data Queries

The large data queries were designed to test the federation engines for real large data use cases, particularly in life sciences domain. These queries span over large datasets (such as Linked TCGA-E, Linked TCGA-M) and involve processing large intermediate result sets (usually in hundreds of thousands, see mean triple pattern selectivities in Figure 3b) or lead to large result sets (minimum 80459, see Figure 3a) and large number of endpoint requests (see Table 9). Consequently, we will see in the evaluation that the query processing time for large data queries exceeds one hour. In order to collect real queries with these characteristics, we contacted different domain experts and obtained a total of 8 large data queries to be included in our benchmark.

4.3. Performance Metrics

As discussed in Section 2, previous works [14, 15, 13, 17] suggest that the following six metrics are important to evaluate the performance of federation engines: (1) the total number of triple pattern-wise (TPW) sources selected during the source selection, (2) the total number of SPARQL ASK requests submitted to perform (1), (3) the completeness (recall) and correctness (precision) of the query result set retrieved, (4) the average source selection time, (5) the average query execution time, (6) the number of endpoint requests. In addition, we also show the

Table 4: SPARQL endpoints specification used in evaluation

Server	CPU(GHz)	RAM	Hard Disk	Server	CPU(GHz)	RAM	Hard Disk
Linked TCGA-M	3.6, i7	32GB	2 TB	Jamendo	2.53, i5	4 GB	300 GB
Linked TCGA-E	3.6, i7	32GB	2 TB	KEGG	2.53, i5	4 GB	500 GB
Linked TCGA-A	2.8, i7	8GB	500 GB	Linked MDB	2.53, i5	4 GB	300 GB
ChEBI	2.53, i5	4 GB	300 GB	New York Times	2.53, i5	4 GB	300 GB
DBpedia-Subset	2.9, i7	8 GB	500 GB	SW Dog Food	2.53, i5	4 GB	300 GB
DrugBank	2.53, i5	4 GB	300 GB	Affymetrix	2.9, i7	8 GB	500 GB
Geo Names	2.9, i7	8 GB	450 GB				

results of the data sources index/data summaries generation time and index compression ratio (i.e., index to dataset ratio). However, they are not applicable to index-free approaches such as FedX [17]. Previous works [13, 15] show that an overestimation of triple pattern-wise sources selected can greatly increase the overall query execution time. This is because extra network traffic is generated and unnecessary intermediate results are retrieved, which are excluded after performing all the joins between query triple patterns. The time consumed by the SPARQL ASK queries during the source selection is directly added into the source selection time, which in turn is added into the overall query execution time.

4.4. Benchmark Usage

One of the key requirements when designing any benchmark is the ease of use in terms of data availability, setting up the evaluation framework, and generating results of the benchmark metrics. To this end, our benchmark homepage contains portable Virtuoso (version 7.10) SPARQL endpoints (both Windows- and Linux-based) which can be started by using one click utilities[19] provided with all SPARQL endpoint downloads. In addition, we provide data dumps of all the datasets used in our benchmark. These data dumps can be uploaded into any other RDF triple store such as Sesame and Fuseki. The total number of TPW sources selected during the source selection as well as the source selection time can only be computed if the source code of the underlying federation engine is available. However, some of the federation engines, e.g., ANAPSID, automatically print the source selection time along with other statistics (e.g., result size, query planning time etc.) after the query execution is completed. We used virtuoso SPARQL endpoints and enabled the http log caching, thus all of the endpoint requests were stored in the endpoints query log file. We calculated the total number of endpoint requests by using a simple java program which reads each endpoints log file line by line and counts the total number of lines (i.e., in all log files from 13 endpoints). Similarly, the total number of SPARQL ASK requests can also be calculated in the same way from the endpoints log file. We provide a Java utility[20] to measure the completeness and correctness of the result set in terms of precision, recall and F1 measure. This utility is generic and can be used for any other benchmark.

5. Evaluation

In this section, we evaluate state-of-the-art SPARQL query federation systems by using both SPARQL 1.0 and SPARQL 1.1 versions of LargeRDFBench queries. We first describe our experimental setup in detail. Then, we present our evaluation results. All data used in this evaluation can be found on the benchmark homepage.

5.1. Experimental Setup

Each of the 13 Virtuoso SPARQL endpoints used in our experiments was installed on a separate machine. The specification of each of the machines is given in Table 4. To avoid server bottlenecks, we started the two largest endpoints (i.e., Linked TCGA-E and Linked TCGA-M) in machines with high processing capabilities. To minimise the network latency we used a dedicated local network. We conducted our experiments on local copies of Virtuoso (version 7.1) SPARQL endpoints with number of buffers 1360000, maximum dirty buffers 1000000, number of server threads 20, result set maximum

[19] Details provided at Benchmark homepage

[20] LargeRDFBench utility: `https://goo.gl/ZmtWG2`

rows 100,000,000,000 and maximum SPARQL endpoint query execution time of 6000,000,000 seconds.

All experiments (i.e., the federation engines themselves) were run on a separate Linux machine with a 2.70GHz i7 processor, 8 GB RAM and 500 GB hard disk. We used the default Java Virtual Machine (JVM) initial memory allocation pool (Xms) size of 1024MB and the maximum memory allocation pool (Xmx) size of 4096MB. Each query was executed 10 times and results were averaged. The query timeout was set to 20 min (1.2×10^6 ms) both for simple and complex queries and 2.5 hours (9×10^6 ms) for large data queries. Furthermore, the query runtime results were statistically analyzed using Wilcoxon Signed Rank Test (WSRT), a non-parametric statistical hypothesis test used when comparing two related samples. We chose this test because it is parameter-free and, unlike a t-test, it does not assume a particular error distribution in the data. For all the significance tests, we set the p-value to 0.05.

None of the engines evaluated herein was able to produce results for the large data queries. Moreover, all engines were only able to retrieve results for a single query, i.e., L7. To be able to compare the selected federation engines, we reran the large data queries experiments on a more powerful clustered server with 32 physical CPU cores of 2.10GHz each and a total RAM of 512GB. Each of the 13 Virtuoso SPARQL endpoints used in our experiments was started as a separate instance on the clustered server. The federation engines were also run on the same machine. We set the maximal amount of memory for each of the federation engines to 128GB.

We compared five SPARQL endpoint federation engines (versions available as of October 2015) – FedX [17], SPLENDID [20], ANAPSID [20], FedX+HiBISCuS [15], SPLENDID+HiBISCuS [15] – on all of the 32 benchmark queries. Note that HiBISCuS [15] is only a source selection approach and FedX+HiBISCuS and SPLENDID+HiBISCuS are the HiBISCuS extensions of the FedX and SPLENDID query federation engines, respectively. To the best of our knowledge, the five systems we chose are the most state-of-the-art SPARQL endpoint federation engines [15]. Of all the systems, only ANAPSID and HiBISCuS perform *join-aware* Triple Pattern-Wise Source Selection (TPWSS). The goal of the *join-aware* TPWSS is to select those data sources that actually *contribute* to the final result set of the query. This is because it is possible that a source contributes to the triple pattern but its results may be excluded after performing a join with the results of another triple pattern.

FedX [17] is an index-free SPARQL query federation system which completely relies on SPARQL ASK queries and a cache (which store the most recent ASK request) to perform TPWSS. This query is forwarded to all of the data sources and those sources which pass the SPARQL ASK test are selected. The result of each SPARQL ASK test is then stored in cache to be used in future. Thus before sending a SPARQL ASK request to a particular data source, a cache lookup is performed. A bind (vectored evaluation in nested loop) join is used for the integration of sub-queries results. We consider two setups for FedX. We evaluated both FedX(cold) and FedX(100%) setups of FedX. The former setup displays the characteristics of FedX with its cache empty and the latter means that cache contains all the information necessary for TPWSS. Thus in later setup, no SPARQL ASK request is used for TPWSS. Consequently, the former setup represents the worst case and the later setup represents the best case scenario.

SPLENDID [20] is an index-assisted approach which makes use of VoiD descriptions as index along with SPARQL ASK queries to perform the TPWSS. A SPARQL ASK query is used when either predicate is unbound (e.g., $< s >$?p $< o >$) or any of the subject (e.g., $< s > < p >$?o) or object (e.g., ?s $< p > < o >$) of the triple pattern is bound. Both bind and hash joins are used for integrating the sub-queries result and a dynamic programming strategy [30] is used to optimize the join order of SPARQL basic graph patterns.

ANAPSID [31] is an index-assisted adaptive query engine that adapts its query execution schedulers to the data availability and runtime status of SPARQL endpoints. ANAPSID performs a heuristic-based source selection presented in its extension [21]. The Adaptive Group Join (based on the Symmetric Hash Join and Xjoin operators) and Adaptive Dependent Join (adjoin) which extends the dependent join operator are used for integrating the sub-queries result.

HiBISCuS [15] is the index-assisted hyper graph based triple pattern-wise source selection approach for SPARQL endpoint federation systems. It intelligently makes use of the hypergraph representation of SPARQL queries and URI's authorities[21] to perform TPWSS. FedX+HiBISCuS, SPLENDID+HiBISCuS are the HiBISCuS extensions of the FedX and

[21] URI syntax: `http://tools.ietf.org/html/rfc3986`

Table 5: Comparison of index construction time, compression ratio, and support for index update. (NA = Not Applicable).

	FedX	SPLENDID	ANAPSID	HiBISCuS
Index Gen. Time(min)	NA	190	6	92
Compression Ratio(%)	NA	99.998	99.999	99.998
Index update?	NA	✗	✗	✓

SPLENDID, respectively. In both the extensions only the source selection is replaced with HiBISCuS. The underlying query execution plan and join order optimization remain the same. The evaluation results show that the performance of both of the original systems are improved with the HiBISCuS extension [15].

5.2. SPARQL 1.0 Experimental Results

5.2.1. Index Construction Time and Compression Ratio

Table 5 shows a comparison of the index/data summaries construction time and the compression ratio[22] of the selected approaches. A high compression ratio is essential for fast index lookups during source selection and query planning. FedX does not rely on an index and makes use of a combination of SPARQL ASK queries and caching to perform the whole of the source selection steps it requires to answer a query. Therefore, these two metrics are not applicable for FedX. As pointed out in [15], ANAPSID only stores the set of distinct predicates corresponding to each data source. Therefore, its index generation time and compression ratio are better than that of HiBISCuS and SPLENDID on our benchmark.

5.2.2. Efficiency of Source Selection

We define efficient source selection in terms of: (1) the total number of triple pattern-wise sources selected (#T), (2) the total number of SPARQL ASK requests (#AR) used to obtain (1), and (3) the source selection time (SST). Table 6 shows the results of these three metrics for the selected approaches. The optimal number of sources were calculated by looking manually into the intermediate results for relevant sources and selecting those sources which contribute to the final result set.

Overall, ANAPSID is the most efficient approach in terms of total TPW sources selected, HiBISCuS is the most efficient in terms of total number of SPARQL ASK requests used, and FedX (100% cached) is the fastest in terms of source selection time (see Table 6). It is important to note that FedX(100% cached) means that the complete source selection is performed by using only cache, i.e., no SPARQL ASK request is used. This the best-case scenario for FedX and very rare in practical cases. Still, FedX (100% cached) clearly overestimates the set of capable sources by more than half to the optimal (474 in FedX vs. 229 optimal). FedX (100% cached) is clearly outperformed by ANAPSID (255 sources selected in total) and HiBISCuS (302 sources selected in total). FedX (100% cached)'s poorer performance is due to FedX only performing TPWSS while both HiBISCuS and ANAPSID perform *join-aware* TPWSS. As mentioned before, such overestimation of sources can be very costly because of the extra network traffic and irrelevant intermediate results retrieval. The effect of such overestimation is consequently even more critical while dealing with large data queries. HiBISCuS is better than ANAPSID in terms of total TPW sources selected both for simple (78 for HiBISCuS and 80 for ANAPSID) and complex (106 for HiBISCuS and 111 for ANAPSID) queries. For large data queries (118 for HiBISCuS and 64 for ANAPSID), HiBISCuS is not able to skip many sources. This is because the approach makes use of the different URI authorities to perform source pruning [15]. However, most of the large data queries come from Linked TCGA with single URI authority (i.e., `tcga.deri.ie`). Hence, HiBISCuS tends to overestimate the number of sources in this case. On the other hand, ANAPSID makes use of SPARQL ASK requests combined with SSGM (Star Shaped Group Multiple Endpoints) [21] to skip a large number of sources. However, SPARQL ASK queries are expensive compared to local index lookups, as performed in HiBISCuS.

5.2.3. Completeness and Correctness of Result Sets

Two systems can only be compared to each other if they provide the same results for a given query execution. Table 7 shows the federation engines and the corresponding LargeRDFBench queries for which

[22] Compression ratio = 100*(1 - index size/total data dump size)

Table 6: Comparison of the source selection in terms of total triple pattern-wise sources selected **#T**, total number of SPARQL ASK requests **#AR**, and source selection time **SST** in msec. **SST*** represents the source selection time for FedX(100% cached i.e. #A =0 for all queries). For ANAPSID, SST represents the query decomposition time. (**T/A** = Total/Avg., where Total is for #T, #AR, and Avg. is SST, SST*). Where Total shows the sum across the individual queries category (i.e., simple, complex, and large data). Net Total shows the overall sum of the values across all 32 queries. The same explanation goes for the average and net average values.

	FedX				SPLENDID			ANAPSID			HiBISCuS			Optimal
Query	#T	#AR	SST	SST*	#T	#AR	SST	#T	#AR	SST	#T	#AR	SST	#T
S1	15	39	238	**5**	15	39	622	**3**	**23**	227	4	26	322	3
S2	**3**	39	229	**6**	**3**	26	380	**3**	**1**	46	**3**	13	201	3
S3	12	65	275	**5**	12	26	358	**5**	2	70	**5**	**0**	52	5
S4	19	65	270	**7**	19	13	340	**5**	3	74	**5**	**0**	130	5
S5	11	52	268	**8**	11	13	330	**4**	1	65	**4**	**0**	90	4
S6	9	52	245	**5**	9	13	303	9	10	197	**8**	**0**	96	8
S7	13	52	248	**6**	13	13	354	**6**	5	273	**6**	**0**	149	6
S8	**1**	26	223	**5**	**1**	**0**	189	**1**	**0**	51	**1**	**0**	9	1
S9	15	39	240	**6**	15	39	592	15	23	356	**9**	**26**	449	3
S10	12	65	296	**5**	12	13	334	**5**	16	262	**5**	**0**	250	5
S11	7	91	300	**7**	7	26	299	**7**	0	333	**7**	**0**	12	7
S12	10	78	260	**5**	10	13	355	**7**	4	105	8	**0**	115	6
S13	9	65	262	**3**	9	26	262	**5**	24	180	7	**0**	132	5
S14	6	65	268	**5**	6	13	252	**5**	2	81	6	**0**	94	7
T/A	142	793	258	**5**	142	273	355	80	114	165	**78**	65	**150**	**67**
C1	11	104	308	**7**	11	13	291	**8**	1	72	9	**0**	120	8
C2	11	104	307	**6**	11	13	347	**8**	2	180	9	**0**	23	8
C3	21	104	318	**5**	21	26	350	**10**	33	549	11	**0**	230	10
C4	28	156	360	**7**	28	**0**	230	28	32	451	**18**	**0**	45	18
C5	33	104	315	**6**	33	**0**	199	**8**	3	156	10	**0**	56	8
C6	24	117	430	**5**	24	**0**	245	**9**	3	90	**9**	**0**	450	9
C7	17	117	436	**7**	17	26	422	**9**	5	380	**9**	**0**	168	9
C8	25	143	402	**4**	25	13	300	**11**	2	308	**11**	**0**	200	11
C9	16	117	400	**6**	16	26	480	**9**	16	185	**9**	**0**	180	9
C10	13	130	350	**8**	13	**0**	240	**11**	6	160	**11**	**0**	150	11
T/A	199	1196	363	**6**	199	117	310	111	103	253	**106**	**0**	**162**	**101**
L1	14	78	282	**5**	14	52	720	**6**	10	260	14	**0**	124	6
L2	10	78	279	**7**	10	13	230	**6**	5	142	10	**0**	94	6
L3	10	91	314	**9**	10	26	314	**7**	5	146	11	**0**	99	7
L4	18	104	321	**7**	18	**0**	198	**8**	8	338	16	**0**	80	8
L5	21	143	400	**5**	21	26	277	**12**	31	10255	20	**0**	130	11
L6	20	130	419	**4**	20	26	298	**10**	52	13173	18	**0**	160	10
L7	20	65	320	**6**	20	13	240	**6**	7	1822	9	**0**	270	5
L8	20	104	366	**7**	20	52	700	**9**	17	404	20	**0**	170	8
T/A	133	793	337	**6**	133	208	372	**64**	135	3317	118	**0**	**140**	**61**
Net T/A	474	2782	311	**6**	474	598	345	**255**	352	980	302	**65**	**151**	**229**

complete and correct results were not retrieved by at least one of the systems. It is important to note that the correct results for all benchmark queries were obtained by loading all the data sources into a single virtuoso triple store and executing the query (a no more federated query) over it. We have not included L8 since every system either timed out or resulted in runtime error, hence the results completeness and correctness cannot be determined in this case. The precision, recall and F1 measures are calculated using the LargeRDFBench utility provided at the aforementioned project home page. Interestingly, none of the systems is able to provide complete and correct results. The incomplete results generated by federation systems can be due to a number of reasons, e.g., their join implementation, the type of network [14], the use of an outdated index or cache or even endpoint restrictions on the maximum result set sizes. However, in our evaluation we always used an up-to-date index and cache, there was no

Table 7: Result set completeness and correctness: Systems with incomplete precision and recall. The values inside brackets show the LargeRDFBench query version, i.e.,SPARQL 1.1. For the rest of the LargeRDFBench queries, all systems give correct and complete results, hence are not shown in the table. (**RE** = Runtime erro, **TO** = Time out)

System	FedX			SPLENDID			ANAPSID			FedX+HiBISCuS			SPLENDID+HiBISCuS		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
C7	0.25	0.19	0.22	1	1	1	1	1	1	0.25	0.19	0.22	1	1	1
S14 (v1.1)	1	0.65	0.79	1	1	1	1	1	1	1	0.65	0.79	1	1	1
C6 (v1.1)	1	0.98	0.99	1	1	1	1	1	1	1	0.98	0.99	1	1	1
L1	TO	TO	TO	1	0.03	0.06	1	0.16	0.28	TO	TO	TO	1	0.03	0.06
L2	0	0	0	TO	TO	TO	TO	TO	TO	0	0	0	1	0.02	0.04
L3	0	0	0	TO	TO	TO	TO	TO	TO	0	0	0	1	1	1
L4	TO	TO	TO	0	0	0	0	0	0	1	0.48	0.65	0	0	0
L5	TO	TO	TO	RE	RE	RE	TO	TO	TO	0	0	0	RE	RE	RE
L6	TO	TO	TO	RE	RE	RE	TO	TO	TO	0	0	0	RE	RE	RE

restriction on SPARQL endpoints maximum result set sizes, and a dedicated local network. Thus, the sole reason (to the best of our knowledge) for the systems at hand not providing complete/correct result is the existence of flaws in the implementation of joins or various SPARQL constructs such as `FILTER`, `REGEX`, etc. For example (as discussed further in the next section), FedX and its HiBISCuS extension possibly give zero results for L2, L3, and L5 due to a flaw in the `FILTER` implementation.

5.2.4. Query Execution Time

The query execution time has often been used as the key metric to compare federation engines. Figure 6, Figure 7, and Table 8 show the query execution time of the selected approaches for simple, complex, and large data queries, respectively. The negligibly small standard deviation error bars (shown on top of each bar) indicate that the data points tend to be very close to the mean, thus suggest a high consistency of the query runtimes in most engines. Note that we considered each time-out to be equal to a runtime of 20min while computing the average runtimes presented in Figure 6 and Figure 7. The query execution time was calculated once all the results were retrieved from the result set iterator. Overall, our results are rather surprising as no system is best over all query types. FedX+HiBISCuS and FedX clearly outperform the remaining systems on *simple queries* (see Figure 6). In particular, FedX and its extension were better than SPLENDID+HiBISCuS in 12/14 queries (11/14 significant improvements on WSRT). On the other hand, SPLENDID+HiBISCuS was better than ANAPSID in 8/14 queries (5/8 significant improvements on WSRT), which in turn was better than SPLENDID in 10/14 queries (9/10 significant improvements on WSRT).

SPLENDID+HiBISCuS performed better than SPLENDID on *complex queries* and was followed by ANAPSID, FedX+HiBISCuS and FedX. ANAPSID is better than SPLENDID+HiBISCuS in 4/7 comparable queries, i.e., on those queries for which complete and correct results are retrieved by both systems. Note that these improvements were all significant according to the WSRT. SPLENDID+HiBISCuS is better than FedX and FedX+HiBISCuS in 5/7 comparable queries (all improvements significant according to WSRT), which is better than SPLENDID in 5/7 comparable queries, with all improvements also being significant (WSRT). We can see that the ranking obtained for complex queries is close to being the reverse of the ranking achieved on simple queries. For example, FedX and its HiBISCuS extension ranked last for complex queries and first in the simple queries. Similarly, SPLENDID ranked last for simple queries and second for complex queries. The results clearly suggest that simple queries benchmarks alone are not sufficient to evaluate the performance of the federation engines. Our results also suggest that smaller source selection time is a key feature of systems which perform well on simple queries as they have smaller execution time. For example, one of the reasons for SPLENDID's poor performance on simple queries is the extra time spent during the source selection (e.g., 5 ms for FedX vs. 355 ms for SPLENDID). This difference of 350 ms is of utmost importance for simple queries given that their execution times are in milliseconds. Both SPLENDID and ANAPSID's performance can be improved by using a source selection cache as used by FedX.

The most important finding for *large data queries* is that no system can be regarded as superior because none can produce complete results for a majority of the queries. This shows that the current *implementation* of query planning strategies (i.e., bushy trees in ANAPSID, left-deep trees in FedX, and dynamic programming [30] in SPLENDID) and

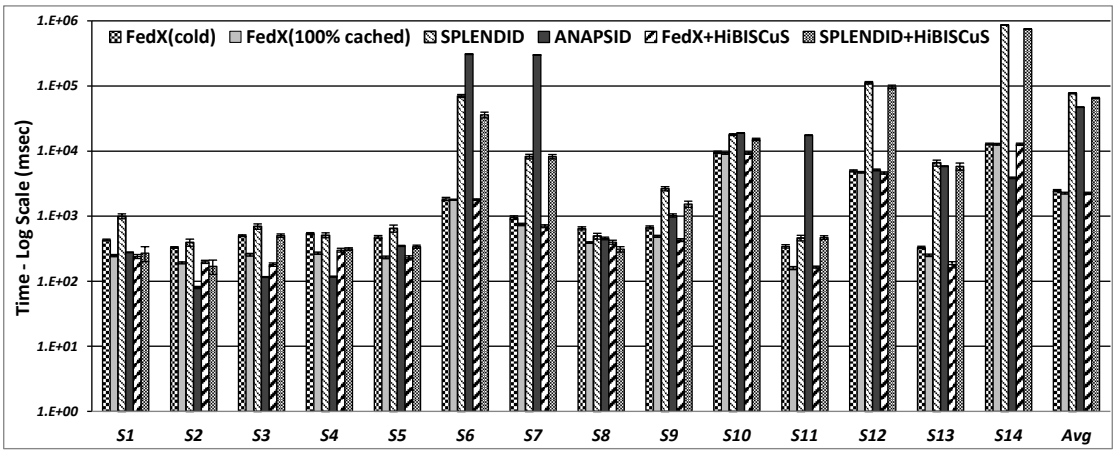


Figure 6: Query execution time for simple category queries.

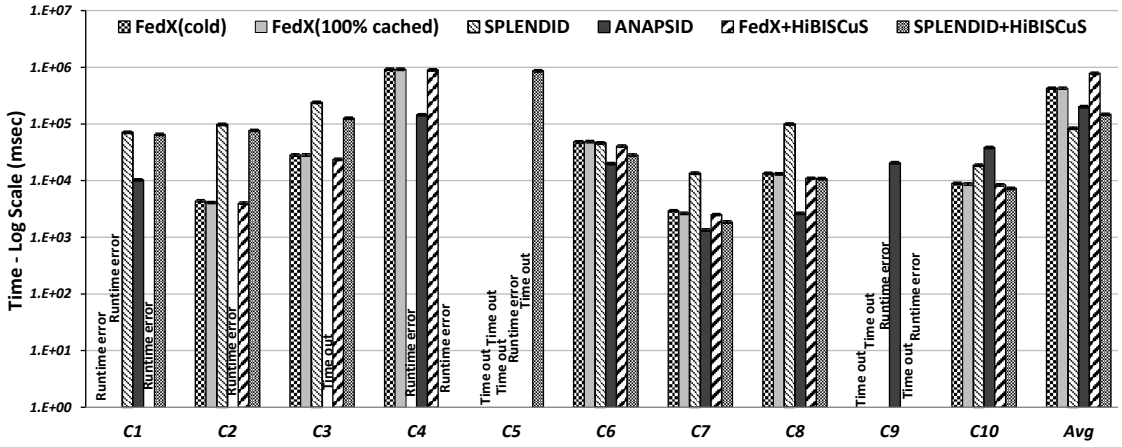


Figure 7: Query execution time for complex category queries.

join techniques (i.e., adaptive group and dependent join in ANAPSID, bind and nested loop in FedX, and bind, hash in SPLENDID) in the selected systems is not mature enough to deal with large data. In addition, we have found that queries terminating within the timeout limit and returning zero results might possibly be caused by a flaw in the `FILTER` implementation. For example, FedX and its HiBISCuS extension give zero results for queries L2, L3, and L5 and send a single endpoint request (ref., 9) for each of these queries. All of these queries contain a `FILTER` clause. However, we found that FedX and its HiBISCuS extension are able to retrieve results by removing the `FILTER` clause and setting the `LIMIT=1` in these queries. We also noticed that for queries with incomplete results (e.g, L1, L4, L8 etc.), FedX and its HiBISCuS extension send a large number of endpoint requests and quickly get some initial results. After that the engines stop sending endpoint requests until the timeout limit is reached. This may be due to some memory leak or possible deadlock in the query execution portion of FedX. Both SPLENDID and SPLENDID+HiBISCuS are able to give complete results for 2/8 large data queries, the highest in comparison to other systems. The query L4 is executed by ANAPSID, SPLENDID, SPLENDID+HiBISCuS within the timeout limit with zero results. In Table 9 we see that ANAPSID sends a total of 11290 requests, SPLENDID sends 16321, and SPLENDID+HiBISCuS send 16220 requests. However, none of these systems is able to find a match for the projection variables. Again this may be because of the possible flaw in `FILTER` or `REGEX` (both of them are used in L4) implementa-

Table 8: Runtimes (in ms) on large data queries with all Virtuoso endpoints. The values inside the brackets show the percentage of the actual query results obtained.(**TO** = Time out after 2.5 hour, **RE** = runtime error).

Qr.	FedX (cold)	FedX (warm)	SPLENDID	ANAPSID	FedX+HiBISCuS	SPLENDID+HiBISCuS
L1	TO (7.2 %)	TO (7.2 %)	123735 (2.73 %)	19672 (15.76 %)	TO (7.2 %)	123700 (2.73 %)
L2	35 (0 %)	35 (0 %)	45473 (1.8 %)	TO (0 %)	76 (0 %)	45479 (1.8 %)
L3	27 (0 %)	27 (0 %)	4877696 (100 %)	TO (0 %)	47 (0 %)	4877991 (100 %)
L4	TO (0.08 %)	TO (0.08 %)	7535531 (0 %)	8775598 (0 %)	62595 (48.34 %)	7535200 (0 %)
L5	TO (0 %)	TO (0 %)	RE (0 %)	TO (0 %)	TO (0 %)	RE (0 %)
L6	TO (0 %)	TO (0 %)	RE (0 %)	TO (0 %)	6127090 (0 %)	RE (0 %)
L7	122633 (100 %)	122500 (100 %)	114456 (100 %)	105447 (100 %)	119449 (100 %)	114400 (100 %)
L8	TO (0.01 %)	TO (0.01 %)	TO (0.05 %)	TO (0.05 %)	TO (0.01 %)	TO (0.05 %)

tions. Note for the same query FedX+HiBISCuS sends 15721 requests to get 48.34% results. The runtime errors thrown by SPLENDID and its HiBISCuS extension for queries L5 and L6 are given in the project website.

While running large data queries, we found that Virtuoso imposes a limit[23] of maximally 2^{20} = 1,048,576 on the maximum number of rows returned as HTTP response. This means that a federation engine based on Virtuoso may end up returning incomplete results if it results for a sub-query with a result set size larger than 1,048,576 rows. To ensure that our results were not tainted by this technical limitation of Virtuoso, we have analyzed all the endpoint requests (given in Table 9) sent by each of the federation engines for each of the LargeRDFBench queries. Our study showed that SPLENDID sends at least one endpoint request with result size greater than 1,048,576 rows. The rest of the federation engines do not send endpoint requests with result size greater than this limit. Given that the endpoints requests with answer set sizes beyond 2^{20} rows were sent exclusively to the 3 Linked TCGA (i.e. Linked TCGA-A, Linked TCGA-E, Linked TCGA-E) datasets, we reran our benchmarking experiments with all federation engines on large data queries (i.e., L1-L8) after replacing the Virtuoso servers for these 3 datasets with FUSEKI[24] servers. Note that the FUSEKI triple store does not have such limit on the maximum number of results returned in response to a query. In this series of experiments, SPLENDID times out with a recall of 0 (i.e., no results generated) for the queries L1, L2 and L4. The other federation engines return results comparable to those presented in Table 8.

[23]This Virtuoso problem is now solved. See `https://github.com/openlink/virtuoso-opensource/issues/700`

[24]FUSEKI: `https://jena.apache.org/documentation/serving_data/`

In a nutshell, our results clearly suggest that benchmarks with only simple queries having small number of result sets are not sufficient to make a fair judgment of the performance of the SPARQL query federation engines. The performance of these systems is greatly affected once the queries go from small to complex and large data. Furthermore, the current state-of-the-art SPARQL query federation systems are not yet ready to deal with large data queries pertaining to real large data use cases.

5.2.5. Number of Endpoint Requests

Table 9 shows the number of endpoint requests submitted by the federation engines during the query execution. Overall, ANAPSID clearly submits fewer requests than the other engines. For simple queries, ANAPSID sends less requests than FedX+HiBISCuS in 8/14 queries, FedX+HiBISCuS is better (in terms of submitting less requests) than FedX in 10/14 queries, FedX is better than SPLENDID+HiBISCuS in 6/11 comparable queries (3 are tied), and SPLENDID+HiBISCuS is better than SPLENDID in 13/14 queries. For complex queries, ANAPSID sends less queries than SPLENDID+HiBISCuS in 4/5 comparable queries, SPLENDID+HiBISCuS is better than FedX+HiBISCuS in 3/6 comparable queries, FedX+HiBISCuS is better than FedX in 5/7 comparable queries, and FedX is better than SPLENDID in 5/6 comparable queries. For large data queries it is hard to compare the federation engines since we were not able to get complete results for the majority of the queries. However, ANAPSID clearly sends fewer requests than the other federation engines.

In summary, we made the following key observations:

- **A smaller number of endpoint requests does not guarantee better query runtime performance:** For example, although ANAPSID is able to send the smallest number of

endpoint requests, it ranks fourth for simple queries and third for complex queries in terms of its query runtime performance. Similarly, the best query runtime engines (i.e., FedX and its extension for simple queries and SPLENDID+HiBISCuS for complex queries) do not send the smallest number of endpoint requests. A careful study of the requests submitted by ANAPSID shows that they are more complex than those submitted by other systems. Consequently, they require more time to be executed by the SPARQL endpoints. Thus, it is also important to consider the complexity of the requests while generating the optimized query execution plans. For example, a request of the type <?s ?p ?o> is more complex than the request of type <s1 p1 ?o>. This also means that the number of endpoint requests is only a good performance metric if the complexity of the endpoint requests is exactly the same or somehow comparable to each other's.

- **The smaller number of triple pattern-wise sources selected generally leads to a smaller number of endpoint requests. However, this may not always be true**: From Table 6 we can see that both FedX+HiBISCuS and SPLENDID+HiBISCuS select fewer numbers of sources in comparison to FedX and SPLENDID. Consequently, in Table 9, we see that these extensions lead to fewer number of endpoint requests in comparison to the original engines. From Table 6, we can see HiBISCuS selects fewer triple pattern-wise selected sources in 24/25 queries compared to both FedX and SPLENDID. Note we count a total of 25 queries instead of the total 32 queries because for 7 queries HiBISCuS, FedX, and SPLENDID select precisely the same number of sources. From Table 9, we can see FedX+HiBISCuS submits fewer requests in 16/19 queries (only considering those queries for which we got complete results) in comparison to FedX. There are 4 (S2, S8, C2, C3) queries for which both FedX and FedX+HiBISCuS submit the same number of endpoint requests. Finally, there are 3 queries (S7, S10, C6) for which FedX+HiBISCuS selects fewer sources, yet submits more endpoint requests than FedX. Similarly, from Table 9, we can see SPLENDID+HiBISCuS submits fewer requests in 20/22 queries (only considering those queries for which we got complete results) in comparison to FedX. For query L3, both SPLENDID+HiBISCuS and SPLENDID submit the same number of requests. Finally, there are 2 queries (S10, L7) for which FedX+HiBISCuS selects fewer sources, yet submits more endpoint requests than SPLENDID. This is because the number of endpoint requests depend upon the query execution plan. Note, as we already discussed, a smaller number of endpoint requests does not guarantee better query runtime performance.

5.3. SPARQL 1.1 Experimental Results

As per current implementations (October 2015), only ANAPSID and FedX support SPARQL 1.1 queries. Thus, they are the only frameworks we compared on the simple and complex SPARQL1.1 queries. For large data queries, the results remained comparable to those presented before. Note that our SPARQL 1.1 version of the queries makes use of the SPARQL `SERVICE` clause, which means the TPWSS is already performed. Furthermore, it is optimally chosen by manually looking at the intermediate results from all the data sources for a given query. Thus, the results presented in Figure 8 show pure query execution performance without the TPWSS.

For simple category queries, ANAPSID is better than FedX in 8/14 queries (all significant improvements on WSRT) in contrast to the results on SPARQL1.0 queries. A deeper look into the results shows the reason for ANAPSID's poor performance on SPARQL 1.0 simple queries was due to the time consumed by the source selection. On average, FedX (100% cached) spent only 6ms for the source selection. On the other hand, ANAPSID spent 165ms on average. For 4/14 queries, ANAPSID's source selection time was greater than the rest of the query execution time (excluding source selection). This shows that efficient TPWSS and the corresponding source selection time is of significant importance while dealing with simple queries. In the simple queries category, FedX overestimates more than half (142 FedX vs. 67 optimal ref. Table 6) of the sources on average. Thus, by using a perfect TPWSS (i.e., in SPARQL 1.1 version), FedX's performance is improved by 54%. This further shows that the *total triple pattern-wise sources selected* is one of the key performance metrics missing in state-of-the-art SPARQL query federation benchmarks. Even though ANAPSID does not substantially overestimate the relevant sources (i.e., 80 ANAPSID vs.

Table 9: Comparison of the average (over 5 runs) number of endpoint requests submitted during the query execution. The case of incomplete results, values inside brackets show the percentage of the actual query results obtained. **Total** shows the sum of the endpoint requests across the individual queries category (i.e., simple, complex, and large data). **Net Total** shows the overall (across all 32 queries) number of endpoint requests submitted by the different systems. (**TO** = Time out after 20 min for complex queries and 2.5 hours for large data queries, **ZRE** = zero results with runtime error, **ZRT** = zero results after the timeout limit)

Query	FedX	SPLENDID	ANAPSID	FedX+HiBISCuS	SPLENDID+HiBISCuS
S1	15	46	25	4	4
S2	2	12	3	2	2
S3	34	23	4	10	11
S4	59	15	5	3	3
S5	27	18	3	6	7
S6	276	3694	19	186	2458
S7	341	495	759	2881	482
S8	1	2	1	1	1
S9	53	142	38	29	65
S10	3527	4361	452	4335	71020
S11	3	10	6	2	3
S12	639	7442	400	458	6539
S13	50909	380	21	50896	315
S14	1961	6509	233	1960	6503
Total	57847	23149	1969	60773	87413
C1	1991	5235	26	1191	3147
C2	549	6649	266 (ZRE)	549	5222
C3	6697	13854	1350 (ZRT)	6697	3299
C4	120892	78 (ZRE)	454	370	0 (ZRE)
C5	19450 (ZRT)	311578 (2.2%)	0 (ZRE)	3568 (ZRT)	500
C6	12386	20	274	15738	2
C7	572 (22 %)	761	67	78 (22 %)	78
C8	1290	5871	108	613	609
C9	2487 (ZRT)	5392 (ZRE)	581	183 (ZRT)	37211 (ZRE)
C10	953	984	278	892	901
Total	167267	350422	3404	29879	50969
L1	2320 (7.2 %)	16 (2.73 %)	1947 (15.76 %)	2320 (7.2 %)	16 (2.73 %)
L2	1 (0 %)	80 (1.8 %)	1609 (ZRT)	1 (0 %)	80 (1.80 %)
L3	1 (0 %)	2734572	5553 (ZRT)	1 (0 %)	2734572
L4	3967 (0.08 %)	16321 (0 %)	11290 (0 %)	15721 (48.34 %)	16220 (0 %)
L5	1 (0 %)	28342 (ZRE)	3840 (ZRT)	1 (0 %)	28212 (ZRE)
L6	3830809 (ZRT)	61810 (ZRE)	1707 (ZRT)	3414400 (0 %)	61419 (ZRE)
L7	74387	867628	267	23381	1341384
L8	206859 (0.01 %)	2423783 (0.05 %)	17302 (0.05 %)	206859 (0.01 %)	2423783 (0.05 %)
Total	4118345	6132552	43515	3662684	6605686
Net Total	4343459	6506123	48888	3753336	6744068

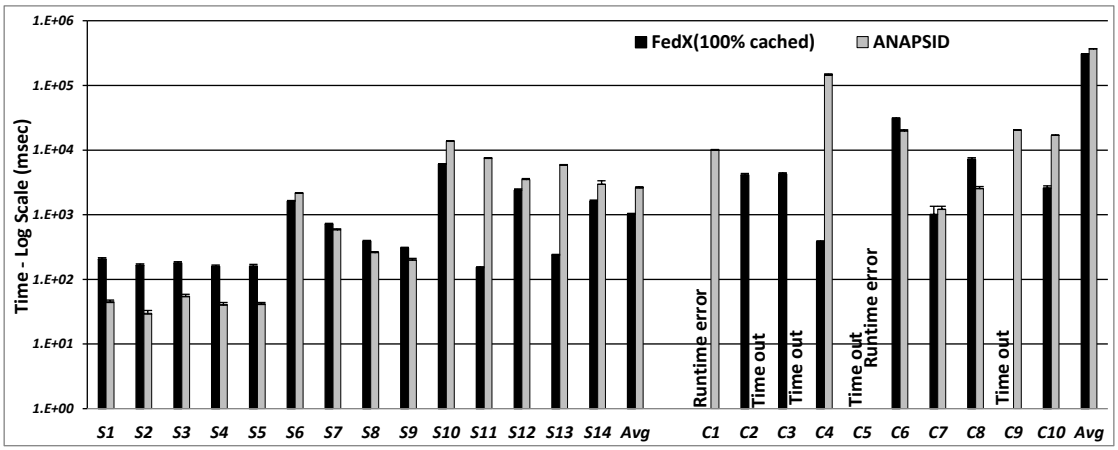


Figure 8: Query execution time for the simple and complex SPARQL 1.1 category queries of LargeRDFBench.

67 optimal), its performance is improved by 94% on SPARQL 1.1 versions of simple queries. The reason for this is the poor query decomposition plan[25] generated for the SPARQL 1.0 version of queries S6 (308514 ms vs. 1620 ms) and S7 (298954 ms vs. 2157 ms). For example, by looking at the query decomposition plan of the SPARQL 1.0 version of the query S6, ANAPSID is not able able to group more than one triple pattern into a single `SERVICE` clause. This means that no join is migrated to the endpoints and the complete results of the individual triple patterns need to be integrated by performing local joins. The total number of `SERVICES` used in the query decomposition plan is 9. On the other hand, for the SPARQL 1.1 version of the same query, ANAPSID is able to group more than one triple pattern into a single `SERVICE` clause, thus more joins are migrated into the endpoints. The number of `SERVICES` used in the query decomposition plan is reduced from 9 to 4. The same explanation goes for S7.

For complex queries the ranking is reversed and FedX is better than ANAPSID in 6/8 comparable queries (all significant improvements on WSRT). This result is expected because FedX overestimated more sources than ANAPSID (199 FedX vs. 111 ANAPSID ref. Table 6). Thus, an optimal TPWSS (as in SPARQL 1.1 version of LargeRDFBench) provides more benefits to FedX. Through optimal source selection, FedX's performance is improved by 28.5% while ANAPSID's performance is only improved by 0.8%.

[25]Decomposed plans given at: `http://goo.gl/AUa0uS`

In general, the results above clearly suggest that FedX's performance can be improved significantly by using smart source selection such as join-aware triple pattern-wise source selections as implemented by HiBISCuS and ANAPSID. Furthermore, the metrics *total triple pattern-wise sources selected* and the corresponding *source selection time*, which were previously ignored, have a significant impact on overall query performance and provide tool developers more fine-grained insights pertaining to their frameworks.

5.3.1. Effect of SPARQL Features on Runtime Performance

As mentioned before, previous works [1, 16] pointed out that the SPARQL benchmark should consider the following *query characteristics*: number of triple patterns, number of join vertices, mean join vertex degree, number of sources span, query result set sizes, mean triple pattern selectivities, BGP-restricted triple pattern selectivity, join-restricted triple pattern selectivity, join vertex types ('star', 'path', 'hybrid', 'sink') etc. In this section, we want to measure the impact of these features on the query runtime performance of the selected engines. To this end, we calculated the Spearman's correlation of the said query features with the query runtime as shown in Table 10. We chose Spearman correlation because it is free of assumptions pertaining to the type of correlation and can hence detect non-linear correlations (in contrast to the Pearson correlation for example, which is designed to detect linear dependencies). We can see that BGP-restricted and join-restricted triple pattern selectivities have a negative correlation with the query execution time, i.e.,

Table 10: Spearman's rank correlation coefficient values to show the correlation of the query execution time and the SPARQL query features. *Results are significant at 1% level (i.e., p<0.01), **Results are significant at 5% level (i.e., p<0.05), ***Results are significant at 10% level (i.e., p<0.10). (**MJVD = Mean Join Vertex Degree, MTPS = Mean Triple Pattern Selectivity, MBRTPS = Mean BGP-restricted Triple Pattern Selectivity, MJRTPS = Mean Join-restricted Triple Pattern Selectivity)**

Query Feature	FedX	SPLENDID	ANAPSID	FedX+HiBISCuS	SPLENDID+HiBISCuS
#Triple Patterns	0.654*	0.536*	0.452**	0.620*	0.492*
#Sources Span	0.233	0.232	0.244	0.019	0.290***
#Results	0.582*	0.553*	0.084	0.534*	0.475**
#Join Vertices	0.275***	0.288***	0.214	0.301***	0.283***
MJVD	0.499*	0.210	0.225	0.381**	0.182
MTPS	0.261	0.304***	0.198	0.237	0.262
MBRTPS	-0.064	-0.021	-0.190	-0.014	-0.041
MJRTPS	-0.508*	-0.334***	-0.223	-0.472**	-0.441**

the lower the value of these two query features the higher the query execution time. However, the join-restricted triple pattern selectivity significantly affects the query runtime compared to BGP-restricted triple pattern selectivity. Similarly, we can see the remaining query features have a positive correlation with query runtime, suggesting all of these features have a direct impact on the query runtime, i.e., the higher the value of the query feature the higher the query execution time. We can see that the number of triple patterns in a query has the highest impact on the query runtime (i.e., *the impact is significant at 1% level for the majority of the federation engines). It is followed by the result size, join-restricted triple pattern selectivity, the number of join vertices, the mean join vertices degree, the mean triple pattern selectivity, the number of sources span, and the BGP-restricted triple pattern selectivity. We can also see these query features do not significantly affect the runtime of the ANAPSID engine, suggesting that the engine may perform well in more complex queries. Our evaluation results show that the query runtime performance of ANAPSID was improved in complex queries of our benchmark. Conversely, FedX is mostly affected by these features. As seen in our evaluation, its performance was decreased in complex queries.

6. Conclusion

In this paper we presented LargeRDFBench, the first billion-triple benchmark for federated SPARQL query engines based on real data and real queries. We presented the three different types of queries contained in the benchmark and compared state-of-the-art systems against these queries. Our results suggest that overall join-aware TPWSS (as implemented by HiBISCuS and ANAPSID) is the superior paradigm when performing source selection. Our evaluation clearly indicates that benchmarks with only simple queries are not sufficient to make a fair judgment of the performance of the SPARQL query federation engines. In addition, while current systems can deal with simple and complex queries, they are currently not up to the challenge of dealing with real large data queries that involve processing large intermediate result sets or lead to large result sets. Furthermore, it is not sufficient to test the federation systems with benchmarks containing only simple queries. Alarmingly, the systems return partly incomplete results without making the user aware of this incompleteness. It is important to mention that we are not claiming that we came up with a perfect benchmark, which might not be possible. However, we have addressed some of the key flaws in existing federation benchmarks which allow us to show that the system rankings achieved via previous federated benchmarks are not the same when dealing with more complex and large data queries.

In the future, triple stores that support SPARQL 1.1 federated queries can also be tested with this benchmark. Furthermore, other metrics such as network latency, number of endpoint requests, and the number of intermediate results can also be measured to provide a more fine-grained evaluation. We have collected a total of 15,287 real SPARQL SERVICE queries[26] from the Bio2RDF query log and generated federated benchmarks of various sizes (ranging

[26] Available at `https://goo.gl/TiuZUY`.

from 15 to 500 queries)[27] using the FEASIBLE [9] benchmark generation framework. This is to further evaluate the restrictions of current federation systems w.r.t. the real complex and large queries that they will be faced with. A discussion of these benchmarks and the corresponding results will be carried out in future work. We hope that this benchmark will further lead to the development of systems that are fit for the current and future developments on the Web.

7. Acknowledgement

This work was supported by grants from the EU H2020 Framework Programme provided for the project HOBBIT (GA no. 688227) and the BMWi project SAKE. We are especially thankful to Helena Deus (Foundations Medicine, Cambridge, MA, USA) and Shanmukha Sampath (Democritus University of Thrace, Alexandroupoli, Greece) for providing real use case large data queries.

References

[1] G. Aluç, O. Hartig, M. T. Özsu, K. Daudjee, Diversified stress testing of rdf data management systems, in: The Semantic Web–ISWC 2014, Springer, 2014, pp. 197–212.

[2] C. Bizer, A. Schultz, The Berlin SPARQL Benchmark, in: International Journal on Semantic Web and Information Systems (IJSWIS), Vol. 5, IGI Global, 2009, pp. 1–24.

[3] Y. Guo, Z. Pan, J. Heflin, LUBM: A benchmark for OWL knowledge base systems, in: Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 3, Elsevier, 2005, pp. 158–182.

[4] M. Morsey, J. Lehmann, S. Auer, A.-C. Ngonga Ngomo, DBpedia SPARQL benchmark - Performance Assessment with Real Queries on Real Data, in: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.), International Semantic Web Conference (ISWC2011), Part I. LNCS, Vol. 7031, Springer Heidelberg, 2011, pp. 454–469.

[5] M. Schmidt, O. Grlitz, P. Haase, G. Ladwig, A. Schwarte, T. Tran, FedBench: A Benchmark Suite for Federated Semantic Data Query Processing, in: L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy, E. Blomqvist (Eds.), The Semantic Web ISWC 2011, Vol. 7031 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 585–600. doi:10.1007/978-3-642-25073-6_37.
URL http://dx.doi.org/10.1007/978-3-642-25073-6_37

[6] M. Schmidt, T. Hornung, G. Lausen, C. Pinkel, Sp2bench: a sparql performance benchmark, in: Internation Conference Data Engineering, 2009.

[7] H. Wu, T. Fujiwara, Y. Yamamoto, J. Bolleman, A. Yamaguchi, Biobenchmark toyama 2012: an evaluation of the performance of triple stores on biological data, Journal of biomedical semantics 5 (1) (2014) 1.

[8] S. Duan, A. Kementsietsidis, K. Srinivas, O. Udrea, Apples and oranges: a comparison of rdf benchmarks and real rdf datasets, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, ACM, 2011, pp. 145–156.

[9] M. Saleem, Q. Mehmood, A.-C. N. Ngomo, Feasible: A feature-based sparql benchmark generation framework, in: The Semantic Web-ISWC 2015, Springer, 2015, pp. 52–69.

[10] M. Arias, J. D. Fernández, M. A. Martínez-Prieto, P. de la Fuente, An empirical study of real-world SPARQL queries, CoRR, 2011.

[11] F. Picalausa, S. Vansummeren, What are real sparql queries like?, in: Proceedings of the International Workshop on Semantic Web Information Management, ACM, 2011, p. 7.

[12] M. Saleem, M. I. Ali, A. Hogan, Q. Mehmood, A.-C. N. Ngomo, Lsq: The linked sparql queries dataset, in: The Semantic Web-ISWC 2015, Springer, 2015, pp. 261–269.

[13] M. Saleem, Y. Khan, A. Hasnain, I. Ermilov, A.-C. Ngonga Ngomo, A fine-grained evaluation of sparql endpoint federation systems, Semantic Web (2015) 1–26.

[14] G. Montoya, M.-E. Vidal, O. Corcho, E. Ruckhaus, C. Buil-Aranda, Benchmarking Federated SPARQL Query Engines: Are Existing Testbeds Enough?, in: P. Cudre Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J.X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, E. Blomqvist, editors, The Semantic Web – ISWC 2012, Part II. LNCS, Vol. 7650, Springer Heidelberg, 2012, pp. 313–324.

[15] M. Saleem, A.-C. Ngonga Ngomo, HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation, in: V. Presutti, C. dAmato, F. Gandon, M. dAquin, S. Staab, A. Tordai (Eds.), The Semantic Web: Trends and Challenges, Vol. 8465 of Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 176–191. doi:10.1007/978-3-319-07443-6_13.
URL http://dx.doi.org/10.1007/978-3-319-07443-6_13

[16] O. Görlitz, M. Thimm, S. Staab, Splodge: systematic generation of sparql benchmark queries for linked open data, in: The Semantic Web–ISWC 2012, Springer, 2012, pp. 116–132.

[17] A. Schwarte, P. Haase, K. Hose, R. Schenkel, M. Schmidt, FedX: Optimization Techniques for Federated Query Processing on Linked Data, in: L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy, E. Blomqvist (Eds.), The Semantic Web ISWC 2011, Vol. 7031 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 601–616. doi:10.1007/978-3-642-25073-6_38.
URL http://dx.doi.org/10.1007/978-3-642-25073-6_38

[18] M. Arenas, C. Gutierrez, J. Pérez, On the Semantics of SPARQL, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 281–307. doi:10.1007/978-3-642-04329-1_13.
URL http://dx.doi.org/10.1007/978-3-642-04329-1_13

[19] O. Hartig, An Overview on Execution Strategies for

[27] Available at https://goo.gl/PsR1wr.

Linked Data Queries, in: Datenbank-Spektrum, Vol. 13, Springer, 2013, pp. 89–99.

[20] O. Görlitz, S. Staab, 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, 2011.

[21] G. Montoya, M.-E. Vidal, M. Acosta, A Heuristic-Based Approach for Planning Federated SPARQL Queries, in: J. F. Sequeda, A. Harth, and O. Hartig, editors, 3rd International Workshop on Consuming Linked Data (COLD 2012) in CEUR Workshop Proceedings, Vol. 905, 2012.

[22] M. Saleem, A.-C. Ngonga Ngomo, J. Xavier Parreira, H. Deus, M. Hauswirth, DAW: Duplicate-AWare Federated Query Processing over the Web of Data, in: H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J. Parreira, L. Aroyo, N. Noy, C. Welty, K. Janowicz (Eds.), The Semantic Web ISWC 2013, Vol. 8218 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 574–590. doi:10.1007/978-3-642-41335-3_36.
URL http://dx.doi.org/10.1007/978-3-642-41335-3_36

[23] Y. Khan, M. Saleem, A. Iqbal, M. Mehdi, A. Hogan, P. Hasapis, A.-C. N. Ngomo, S. Decker, R. Sahay, SAFE: Policy Aware SPARQL Query Federation Over RDF Data Cubes, in: A. Paschke, A. Burger, P. Romano, M. S. Marshall, A. Splendiani, editors, Proceedings of the 7th International Workshop on Semantic Web Applications and Tools for Life Sciences in CEUR Workshop Proceedings, Vol. 1320, 2014.

[24] Y. Khan, M. Saleem, M. Mehdi, A. Hogan, Q. Mehmood, D. Rebholz-Schuhmann, R. Sahay, Safe: Sparql federation over rdf data cubes with access control, Journal of Biomedical Semantics 8 (1). doi:10.1186/s13326-017-0112-6.
URL http://dx.doi.org/10.1186/s13326-017-0112-6

[25] A. Hasnain, Q. Mehmood, S. Sana e Zainab, M. Saleem, C. Warren, D. Zehra, S. Decker, D. Rebholz-Schuhmann, Biofed: federated query processing over life sciences linked open data, Journal of Biomedical Semantics 8 (1) (2017) 13. doi:10.1186/s13326-017-0118-0.
URL http://dx.doi.org/10.1186/s13326-017-0118-0

[26] M. Saleem, S. S. Padmanabhuni, A.-C. N. Ngomo, A. Iqbal, J. S. Almeida, S. Decker, H. F. Deus, Topfed: Tcga tailored federated query processing and linking to lod, Journal of biomedical semantics 5 (1) (2014) 1.

[27] M. Saleem, R. Maulik, I. Aftab, S. Shanmukha, H. Deus, A.-C. Ngonga Ngomo, Fostering Serendipity through Big Linked Data, in: Semantic Web Challenge at International Semantic Web Conference, 2013.

[28] M. Saleem, M. R. Kamdar, A. Iqbal, S. Sampath, H. F. Deus, A.-C. N. Ngomo, Big linked cancer data: Integrating linked {TCGA} and pubmed, Web Semantics: Science, Services and Agents on the World Wide Web 2728 (2014) 34 – 41, semantic Web Challenge 2013. doi:http://dx.doi.org/10.1016/j.websem.2014.07.004.
URL http://www.sciencedirect.com/science/article/pii/S1570826814000523

[29] N. Jiang, L. J. Leach, X. Hu, E. Potokina, T. Jia, A. Druka, R. Waugh, M. J. Kearsey, Z. W. Luo, Methods for evaluating gene expression from affymetrix microarray datasets, BMC bioinformatics 9 (1) (2008) 284.

[30] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, T. G. Price, Access Path Selection in a Relational Database Management System, in: Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, SIGMOD '79, ACM, New York, NY, USA, 1979, pp. 23–34. doi:10.1145/582095.582099.
URL http://doi.acm.org/10.1145/582095.582099

[31] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, E. Ruckhaus, ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints, in: L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy, E. Blomqvist (Eds.), The Semantic Web ISWC 2011, Vol. 7031 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 18–34. doi:10.1007/978-3-642-25073-6_2.
URL http://dx.doi.org/10.1007/978-3-642-25073-6_2

Appendix: LargeRDFBench queries

The LargeRDFBench queries are given in the Listing below and can also be downloaded from the project website.

LargeRDFBench queries.

```
PREFIX owl:        <http://www.w3.org/2002/07/owl#>
PREFIX dbpediaR: <http://dbpedia.org/resource>
PREFIX dbpedia:  <http://dbpedia.org/ontology/>
PREFIX nytimes:  <http://data.nytimes.com/elements/>
PREFIX rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX linkedmdb:    <http://data.linkedmdb.org/resource/movie/>
PREFIX purl:     <http://purl.org/dc/elements/1.1/>
PREFIX foaf:             <http://xmlns.com/foaf/0.1/>
PREFIX geonames:         <http://www.geonames.org/ontology#>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX drugs: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/>
PREFIX drugcategory: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugcategory
    />
PREFIX kegg: <http://bio2rdf.org/ns/kegg#>
PREFIX bio2RDF: <http://bio2rdf.org/ns/bio2rdf#>
PREFIX chebi: <http://bio2rdf.org/ns/bio2rdf#>
PREFIX drugtype: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugtype/>
PREFIX mo:               <http://purl.org/ontology/mo/>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix swc: <http://data.semanticweb.org/ns/swc/ontology#>
prefix swrc: <http://swrc.ontoware.org/ontology#>
prefix eswc: <http://data.semanticweb.org/conference/eswc/>
prefix iswc:  <http://data.semanticweb.org/conference/iswc/2009/>
PREFIX tcga: <http://tcga.deri.ie/schema/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX affymetrix: <http://bio2rdf.org/affymetrix_vocabulary:>
############################ S1: SPARQL 1.0 ############################
#Find all information about Barack Obama.
SELECT ?predicate ?object
WHERE
{
   { dbpediaR:Barack_Obama ?predicate ?object }
   UNION
   {
    ?subject owl:sameAs dbpediaR:Barack_Obama .
     ?subject ?predicate ?object
   }
   }
#---------------------- S1: SPARQL 1.1 ----------------------
SELECT ?predicate ?object
WHERE
{
 { SERVICE <dbpedia-subset endpoint> { dbpediaR:Barack_Obama ?predicate ?object } }
   UNION
   {
     SERVICE <newyork-times endpoint> {
     ?subject owl:sameAs dbpediaR:Barack_Obama .
     ?subject ?predicate ?object }
   }
}
```

LargeRDFBench queries.

```
############################ S2: SPARQL 1.0 ############################
#Return Barack Obama's party membership and news pages.
SELECT ?party ?page
WHERE
{
   dbpediaR:Barack_Obama dbpedia:party ?party .
   ?x nytimes:topicPage ?page .
   ?x owl:sameAs dbpediaR:Barack_Obama .
}
#---------------------- S2: SPARQL 1.1 ----------------------
SELECT ?party ?page
WHERE
{
  SERVICE <dbpedia-subset endpoint> {dbpediaR:Barack_Obama dbpedia:party ?party .}
   SERVICE <newyork-times endpoint>
     {
       ?x nytimes:topicPage ?page .
       ?x owl:sameAs dbpediaR:Barack_Obama .
     }
}
############################ S3: SPARQL 1.0 ############################
# Return for all US presidents their party membership and news pages about them.
SELECT ?president ?party ?page
WHERE
{
   ?president rdf:type dbpedia:President .
   ?president dbpedia:nationality dbpediaR:United_States .
   ?president dbpedia:party ?party .
   ?x nytimes:topicPage ?page .
   ?x owl:sameAs ?president .
}
#---------------------- S3: SPARQL 1.1 ----------------------
SELECT ?president ?party ?page
WHERE
{
  SERVICE <dbpedia-subset endpoint>
  {
   ?president rdf:type dbpedia:President .
   ?president dbpedia:nationality dbpediaR:United_States .
   ?president dbpedia:party ?party .
  }
  SERVICE <newyork-times endpoint>
  {
   ?x nytimes:topicPage ?page .
   ?x owl:sameAs ?president .
   }
}
```

LargeRDFBench queries.

```
############################ S4: SPARQL 1.0 ############################
#Find all news about actors starring in a movie with name Tarzan.
SELECT ?actor ?news
WHERE
{
   ?film purl:title 'Tarzan' .
   ?film linkedmdb:actor ?actor .
   ?actor owl:sameAs ?x.
   ?y owl:sameAs ?x .
   ?y nytimes:topicPage ?news
}
#---------------------- S4: SPARQL 1.1 ----------------------
SELECT ?actor ?news
WHERE
{
   SERVICE <linked-mdb endpoint>
  {
   ?film purl:title 'Tarzan' .
   ?film linkedmdb:actor ?actor .
   ?actor owl:sameAs ?x.
   }
   SERVICE <newyork-times endpoint>
  {
   ?y owl:sameAs ?x .
   ?y nytimes:topicPage ?news
   }
}
############################ S5: SPARQL 1.0 ############################
#Find the director and the genre of movies directed by Italians.
SELECT ?film ?director ?genre
WHERE
{
   ?film dbpedia:director  ?director .
   ?director dbpedia:nationality dbpediaR:Italy .
   ?x owl:sameAs ?film .
   ?x linkedmdb:genre ?genre .
}
#---------------------- S5: SPARQL 1.1 ----------------------
SELECT ?film ?director ?genre
WHERE
{
 SERVICE <dbpedia-subset endpoint>
  {
   ?film dbpedia:director  ?director .
   ?director dbpedia:nationality dbpediaR:Italy .
   }
   SERVICE <linked-mdb endpoint>
  {
   ?x owl:sameAs ?film .
   ?x linkedmdb:genre ?genre .
   }
}
```

LargeRDFBench queries.

```
############################ S6: SPARQL 1.0 ############################
#Find all musical artists based in Germany.
SELECT ?name ?location
WHERE
{
   ?artist foaf:name ?name .
   ?artist foaf:based_near ?location .
   ?location geonames:parentFeature ?germany .
   ?germany geonames:name 'Federal_Republic_of_Germany'
}
#---------------------- S6: SPARQL 1.1 ----------------------
SELECT ?name ?location
WHERE {
{
SERVICE <jamendo endpoint>
{
?artist foaf:name ?name .
?artist foaf:based_near ?location .}
}
UNION
{
SERVICE <swdf endpoint>
{
?artist foaf:name ?name .
?artist foaf:based_near ?location .}
}
 {
SERVICE <geonames endpoint> {
 ?location geonames:parentFeature ?germany .
 ?germany geonames:name 'Federal_Republic_of_Germany'
}
}
UNION
{
SERVICE <newyork-times endpoint>
{
 ?location geonames:parentFeature ?germany .
 ?germany geonames:name 'Federal_Republic_of_Germany'
 }
}
}
############################ S7: SPARQL 1.0 ############################
#Find all news about locations in the state of California.
SELECT ?location ?news
WHERE
{
   ?location geoname:parentFeature ?parent .
   ?parent geoname:name 'California' .
   ?y owl:sameAs ?location .
   ?y nytimes:topicPage ?news
}
```

LargeRDFBench queries.

```
#----------------------- S7: SPARQL 1.1 -----------------------
SELECT ?location ?news
WHERE
{
  {
  SERVICE <newyork-times endpoint>
    {
   ?location geoname:parentFeature ?parent .
   ?parent geoname:name 'California' .
     }
   }
   UNION
   {
   SERVICE <geonames endpoint>
    {
   ?location geoname:parentFeature ?parent .
   ?parent geoname:name 'California' .
   }
   }
   SERVICE <newyork-times endpoint>
   {
   ?y owl:sameAs ?location .
   ?y nytimes:topicPage ?news
   }
}
############################ S8: SPARQL 1.0 ############################
#Find all drugs from Drugbank and DBpedia with their melting points.
SELECT ?drug ?melt
WHERE
{
    {
    ?drug drugbank:meltingPoint ?melt.
    }
    UNION
    {
    ?drug drugbank:Drug/meltingPoint ?melt .
    }
}
#----------------------- S8: SPARQL 1.1 -----------------------
SELECT ?drug ?melt
WHERE
{
    {
     SERVICE <drugbank endpoint> {?drug drugbank:meltingPoint ?melt .}
     }
    UNION
    {
     SERVICE <dbpedia-subset endpoint> {?drug drugbank:Drug/meltingPoint ?melt .}
     }
}
```

LargeRDFBench queries.

```
############################ S9: SPARQL 1.0 ############################
#Find all entities from all available databases describing Caffeine.
SELECT ?predicate ?object
WHERE
{
    {
      drugs:DB00201 ?predicate ?object .
     }
    UNION
    {
     drugs:DB00201 owl:sameAs ?caff .
     ?caff ?predicate ?object .
    }
}
#---------------------- S9: SPARQL 1.1 ----------------------
SELECT ?predicate ?object
WHERE
{
    {
      SERVICE <drugbank endpoint> {drugs:DB00201 ?predicate ?object .}
     }
    UNION
    {
    SERVICE <drugbank endpoint> {drugs:DB00201 owl:sameAs ?caff .}
    SERVICE <dbpedia-subset endpoint> {?caff ?predicate ?object .}
    }
}
############################ S10: SPARQL 1.0 ############################
#For all drugs in DBpedia find all drugs they interact with along with an
    explanation of the interaction.
SELECT ?Drug ?IntDrug ?IntEffect
WHERE
{
    ?Drug rdf:type dbpedia:Drug .
    ?y owl:sameAs ?Drug .
    ?Int drugbank:interactionDrug1 ?y .
    ?Int drugbank:interactionDrug2 ?IntDrug .
    ?Int drugbank:text ?IntEffect .
}
#---------------------- S10: SPARQL 1.1 ----------------------
SELECT ?Drug ?IntDrug ?IntEffect
WHERE
{
    SERVICE <dbpedia-subset endpoint> {?Drug rdf:type dbpedia:Drug .}
    SERVICE <drugbank endpoint>
    {
    ?y owl:sameAs ?Drug .
    ?Int drugbank:interactionDrug1 ?y .
    ?Int drugbank:interactionDrug2 ?IntDrug .
    ?Int drugbank:text ?IntEffect .
    }
}
```

LargeRDFBench queries.

```
############################ S11: SPARQL 1.0 ############################
#Find all the equations of reactions related to drugs from category Cathartics and
    their drug description.
SELECT ?drugDesc ?cpd ?equation
WHERE
{
   ?drug drugbank:drugCategory drugcategory:cathartics .
   ?drug drugbank:keggCompoundId ?cpd .
   ?drug drugbank:description ?drugDesc .
   ?enzyme kegg:xSubstrate ?cpd .
   ?enzyme rdf:type kegg:Enzyme .
   ?reaction kegg:xEnzyme ?enzyme .
   ?reaction kegg:equation ?equation .
}
#---------------------- S11: SPARQL 1.1 ----------------------
SELECT ?drugDesc ?cpd ?equation
WHERE
{
 SERVICE <drugbank endpoint>
  {
   ?drug drugbank:drugCategory drugcategory:cathartics .
   ?drug drugbank:keggCompoundId ?cpd .
   ?drug drugbank:description ?drugDesc .
   }
 SERVICE <kegg endpoint>
  {
   ?enzyme kegg:xSubstrate ?cpd .
   ?enzyme rdf:type kegg:Enzyme .
   ?reaction kegg:xEnzyme ?enzyme .
   ?reaction kegg:equation ?equation .
   }
}
############################ S12: SPARQL 1.0 ############################
#Find all drugs from Drugbank together with the URL of the corresponding page stored
     in KEGG and the URL to the image derived from ChEBI.
SELECT ?drug ?keggUrl ?chebiImage
WHERE {
   ?drug rdf:type drugbank:drugs .
   ?drug drugbank:keggCompoundId ?keggDrug .
   ?keggDrug bio2RDF#url ?keggUrl .
   ?drug drugbank:genericName ?drugBankName .
   ?chebiDrug purl:title ?drugBankName .
   ?chebiDrug bio2RDF:image ?chebiImage . }
#---------------------- S12: SPARQL 1.1 ----------------------
SELECT ?drug ?keggUrl ?chebiImage
WHERE {
  SERVICE <drugbank endpoint> {
   ?drug rdf:type drugbank:drugs .
   ?drug drugbank:keggCompoundId ?keggDrug .
   ?drug drugbank:genericName ?drugBankName .}
  SERVICE <kegg endpoint> {?keggDrug bio2RDF#url ?keggUrl .}
  SERVICE <chebi endpoint> {
   ?chebiDrug purl:title ?drugBankName .
   ?chebiDrug bio2RDF:image ?chebiImage .}}
```

LargeRDFBench queries.

```
############################ S13: SPARQL 1.0 ############################
#Find KEGG drug names of all drugs in Drugbank belonging to category Micronutrient.
SELECT ?drug ?title
WHERE {
    ?drug drugbank:drugCategory drugcategory:micronutrient .
    ?drug drugbank:casRegistryNumber ?id .
    ?keggDrug rdf:type kegg:Drug .
    ?keggDrug bio2RDF:Ref ?id .
    ?keggDrug purl:title ?title .
}
#---------------------- S13: SPARQL 1.1 ----------------------
SELECT ?drug ?title
WHERE {
    SERVICE <drugbank endpoint>
    {
    ?drug drugbank:drugCategory drugcategory:micronutrient .
    ?drug drugbank:casRegistryNumber ?id .
    }
    SERVICE <kegg endpoint>
    {
    ?keggDrug rdf:type kegg:Drug .
    ?keggDrug bio2RDF:Ref ?id .
    ?keggDrug purl:title ?title .
    }
}
############################ 14: SPARQL 1.0 ############################
#Find all drugs that affect humans and mammals. For those having a description of
    their biotransformation. Also return this description.
SELECT ?drug ?transform ?mass
WHERE {
 ?drug drugbank:affectedOrganism>  'Humans␣and␣other␣mammals'.
 ?drug drugbank:casRegistryNumber ?cas .
 ?keggDrug bio2RDF:xRef ?cas .
 ?keggDrug bio2RDF:mass ?mass .
 OPTIONAL { ?drug drugbank:biotransformation ?transform . }
}
#---------------------- S14: SPARQL 1.1 ----------------------
SELECT ?drug ?transform ?mass
WHERE {
 SERVICE <drugbank endpoint>
 {
 ?drug drugbank:affectedOrganism>  'Humans␣and␣other␣mammals'.
 ?drug drugbank:casRegistryNumber ?cas .
 }
  SERVICE <kegg endpoint>
 {
 ?keggDrug bio2RDF:xRef ?cas .
 ?keggDrug bio2RDF:mass ?mass .
 }
 OPTIONAL {SERVICE <drugbank endpoint> {?drug drugbank:biotransformation ?transform
     .}
 }
}
```

LargeRDFBench queries.

```
############################ C1: SPARQL 1.0 ############################
#Find the equations of chemical reactions and reaction title related to drugs with
    drug description and drug type 'smallMolecule'. Show only those whose molecular
    weight average larger then 114.
SELECT DISTINCT ?drug    ?drugDesc ?molecularWeightAverage        ?compound    ?
    ReactionTitle     ?ChemicalEquation
WHERE
{
?drug             drugbank:description      ?drugDesc .
?drug             drugbank:drugType         drugtype:smallMolecule .
?drug             drugbank:keggCompoundId ?compound .
?enzyme           kegg:xSubstrate           ?compound .
?Chemicalreaction         kegg:xEnzyme     ?enzyme .
?Chemicalreaction         kegg:equation    ?ChemicalEquation .
?Chemicalreaction         purl:title                ?ReactionTitle
OPTIONAL
 {
  ?drug drugbank:molecularWeightAverage ?molecularWeightAverage .
  FILTER (?molecularWeightAverage > 114)
 }
}
LIMIT 1000
#---------------------- C1: SPARQL 1.1 ----------------------
SELECT DISTINCT ?drug    ?drugDesc ?molecularWeightAverage        ?compound    ?
    ReactionTitle     ?ChemicalEquation
WHERE
{
SERVICE <drugbank endpoint>
{
?drug             drugbank:description      ?drugDesc .
?drug             drugbank:drugType         drugtype:smallMolecule .
?drug             drugbank:keggCompoundId ?compound .
}
SERVICE <kegg endpoint>
{
?enzyme           kegg:xSubstrate           ?compound .
?Chemicalreaction         kegg:xEnzyme     ?enzyme .
?Chemicalreaction         kegg:equation    ?ChemicalEquation .
?Chemicalreaction         purl:title                ?ReactionTitle
}
OPTIONAL
 {
 SERVICE <drugbank endpoint>
 {
  ?drug drugbank:molecularWeightAverage ?molecularWeightAverage .
  FILTER (?molecularWeightAverage > 114)
  }
 }
}
LIMIT 1000
```

LargeRDFBench queries.

```
############################ C2: SPARQL 1.0 ############################
#Find all the drugs with their mass and chebiIupacName optionally the Inchi values
    retrieving from two sources are equal.
SELECT ?drug ?keggmass ?chebiIupacName
WHERE
{
?drug rdf:type drugbank:drugs .
?drug drugbank:keggCompoundId ?keggDrug .
?keggDrug bio2RDF:mass ?keggmass .
?drug drugbank:genericName ?drugBankName .
?chebiDrug purl:title ?drugBankName .
?chebiDrug chebi:iupacName ?chebiIupacName .
OPTIONAL
 {
?drug drugbank:inchiIdentifier ?drugbankInchi .
?chebiDrug bio2RDF:inchi ?chebiInchi.
FILTER (?drugbankInchi = ?chebiInchi)
 }
}
#---------------------- C2: SPARQL 1.1 ----------------------
SELECT ?drug ?keggmass ?chebiIupacName
WHERE {
SERVICE <drugbank endpoint>
{
?drug rdf:type drugbank:drugs .
?drug drugbank:keggCompoundId ?keggDrug .
}
SERVICE <kegg endpoint> {?keggDrug bio2RDF:mass ?keggmass .}
SERVICE <drugbank endpoint> {?drug drugbank:genericName ?drugBankName .}
SERVICE <drugbank endpoint>
{
?chebiDrug purl:title ?drugBankName .
?chebiDrug chebi:iupacName ?chebiIupacName .
}
OPTIONAL  {
SERVICE <drugbank endpoint> {?drug drugbank:inchiIdentifier ?drugbankInchi .}
SERVICE <chebi endpoint> {?chebiDrug bio2RDF:inchi ?chebiInchi .}
FILTER (?drugbankInchi = ?chebiInchi) }
}
############################ C3: SPARQL 1.0 ############################
#Find the names of music artists with the news about locations where these artist
    are based.
SELECT DISTINCT ?artist ?name ?location ?anylocation
WHERE {
        ?artist a mo:MusicArtist ;
         foaf:name ?name ;
                foaf:based_near ?location .
         ?location geonames:parentFeature ?locationName .
         ?locationName geonames:name ?anylocation .
         ?nytLocation owl:sameAs ?location.
         ?nytLocation nytimes:topicPage ?news
OPTIONAL {?locationName geonames:name 'Islamic_Republic_of_Afghanistan' . }
}
```

LargeRDFBench queries.

```
#----------------------- C3: SPARQL 1.1 -----------------------
SELECT DISTINCT ?artist ?name ?location ?anylocation
WHERE {
SERVICE <jamendo endpoint> {
        ?artist a mo:MusicArtist ;
         foaf:name ?name ;
         foaf:based_near ?location . }
{
 SERVICE <geonames endpoint>  {
         ?location geonames:parentFeature ?locationName .
         ?locationName geonames:name ?anylocation . }
 }
 UNION
 {
 SERVICE <newyork-times endpoint>   {
         ?location geonames:parentFeature ?locationName .
         ?locationName geonames:name ?anylocation .   }
 }
   SERVICE <newyork-times endpoint>   {
         ?nytLocation owl:sameAs ?location.
         ?nytLocation nytimes:topicPage ?news  }
OPTIONAL
{
SERVICE <geonames endpoint> {?locationName geonames:name 'Islamic Republic of
    Afghanistan' . }
}
}
############################ C4: SPARQL 1.0 ############################
#Find the country name with its relevant information such as population country code
     etc.
SELECT DISTINCT ?countryName ?countryCode ?locationMap ?population ?longitude ?
    latitude ?nationalAnthem ?foundingDate ?largestCity ?ethnicGroup ?motto
{
?NYTplace geonames:name ?countryName.
?NYTplace geonames:countryCode ?countryCode.
?NYTplace geonames:population  ?population.
?NYTplace geo:long    ?longitude.
?NYTplace geo:lat     ?latitude.
?NYTplace owl:sameAs   ?geonameplace.
OPTIONAL
 {
?geonameplace dbpedia:capital ?capital.
?geonameplace dbpedia:anthem ?nationalAnthem.
?geonameplace dbpedia:foundingDate ?foundingDate.
?geonameplace dbpedia:largestCity ?largestCity.
?geonameplace dbpedia:ethnicGroup ?ethnicGroup.
?geonameplace dbpedia:motto ?motto.
 }
}
LIMIT 50
```

LargeRDFBench queries.

```
#----------------------- C4: SPARQL 1.1 -----------------------
SELECT DISTINCT ?countryName ?countryCode ?locationMap ?population ?longitude ?
    latitude ?nationalAnthem ?foundingDate ?largestCity ?ethnicGroup ?motto
{       {
          SERVICE <newyork-times endpoint>  {
           ?NYTplace geonames:name ?countryName;
                 geonames:countryCode ?countryCode;
                 geonames:population  ?population;
                 geo:long    ?longitude;
                 geo:lat      ?latitude;
                 owl:sameAs    ?geonameplace. }
    }
          UNION
          {
           SERVICE <geonames endpoint> {
           ?NYTplace geonames:name ?countryName;
                 geonames:countryCode ?countryCode;
                 geonames:population  ?population;
                 geo:long    ?longitude;
                 geo:lat      ?latitude;
                 owl:sameAs    ?geonameplace. }
      }
OPTIONAL
{
        SERVICE <http://dbpedia-subset endpoint> {
                 ?geonameplace dbpedia:capital ?capital;
                 dbpedia:anthem ?nationalAnthem;
                 dbpedia:foundingDate ?foundingDate;
                 dbpedia:largestCity ?largestCity;
                 dbpedia:ethnicGroup ?ethnicGroup;
                 dbpedia:motto ?motto. }
}
}
LIMIT 50
############################ C5: SPARQL 1.0 ############################
#The names of the actors, their data of birth, their Spouse name who worked in any
    movie with movie name, its title and date of movie release.
SELECT ?actor ?movie ?movieTitle ?movieDate ?birthDate ?spouseName
{
        ?actor rdfs:label ?actor_name_en;
               dbpedia:birthDate ?birthDate ;
               dbpedia:spouse ?spouseURI .
        ?spouseURI rdfs:label ?spouseName .
        ?imdbactor linkedmdb:actor_name ?actor_name.
        ?movie linkedmdb:actor ?imdbactor ;
             dcterms:title ?movieTitle ;
             dcterms:date ?movieDate
      FILTER(STR(?actor_name_en )= STR(?actor_name))
}
LIMIT 500
```

LargeRDFBench queries.

```
#----------------------- C5: SPARQL 1.1 -----------------------
SELECT ?actor ?movie ?movieTitle ?movieDate ?birthDate ?spouseName
{
        SERVICE <dbpedia-subset endpoint> {
                ?actor rdfs:label ?actor_name_en;
        dbpedia:birthDate ?birthDate ;
                dbpedia:spouse ?spouseURI .
                ?spouseURI rdfs:label ?spouseName . }
        SERVICE <linked-mdb endpoint> {
        ?imdbactor linkedmdb:actor_name ?actor_name.
        ?movie linkedmdb:actor ?imdbactor ;
    dcterms:title ?movieTitle ;
    dcterms:date ?movieDate }
FILTER(STR(?actor_name_en) = STR(?actor_name))
}
LIMIT 500
############################ C6: SPARQL 1.0 ############################
#Find all news, its variants, total number of news articles, when it was first used
    and when it was latestly used, about actors starring in any movie.
SELECT ?actor ?filmTitle ?news ?variants ?articleCount ?first_use ?latest_use
WHERE
{
?film purl:title                          ?filmTitle .
?film linkedmdb:actor                     ?actor .
?actor owl:sameAs                         ?dbpediaURI.
?nytURI owl:sameAs                        ?dbpediaURI .
?nytURI nytimes:topicPage                 ?news ;
        nytimes:number_of_variants        ?variants;
        nytimes:associated_article_count  ?articleCount;
        nytimes:first_use                 ?first_use;
        nytimes:latest_use                ?latest_use
}
ORDER BY (?actor)
#----------------------- C6: SPARQL 1.1 -----------------------
 SELECT ?actor ?filmTitle ?news ?variants ?articleCount ?first_use ?latest_use
 WHERE
 {
        SERVICE <linked-mdb endpoint> {
        ?film purl:title                          ?filmTitle .
        ?film linkedmdb:actor                     ?actor .
        ?actor owl:sameAs                         ?dbpediaURI. }
        SERVICE <newyork-times endpoint> {
        ?nytURI owl:sameAs                        ?dbpediaURI .
        ?nytURI nytimes:topicPage                 ?news ;
        nytimes:number_of_variants        ?variants;
        nytimes:associated_article_count  ?articleCount;
        nytimes:first_use                 ?first_use;
        nytimes:latest_use                ?latest_use }
 }
 ORDER BY (?actor)
```

LargeRDFBench queries.

```
############################ C7: SPARQL 1.0 ############################
#For all the authors and their corresponding publication who also had role in ESWC
    2010, along with the information in which countries these authors lived in with
    the optional information of geographical location of that country and the name
    its capital (if available).
SELECT DISTINCT ?author ?role ?paper ?place ?capital ?latitude ?longitude  ?
    proceedings
WHERE
{
?role swc:isRoleAt  eswc:2010.
?role swc:heldBy ?author .
?proceedings swc:relatedToEvent eswc:2010.
?paper swrc:author ?author .
?author foaf:based_near ?place.
?paper swc:isPartOf ?proceedings .
OPTIONAL
{
?place dbpedia:capital ?capital;
        geo:lat ?latitude;
        geo:long ?longitude.
}
}
#---------------------- C7: SPARQL 1.1 ----------------------
SELECT  DISTINCT ?author ?role ?paper ?place ?capital ?latitude ?longitude  ?
    proceedings WHERE
{
        SERVICE <swdf endpoint>
        {
        ?role swc:isRoleAt  eswc:2010.
        ?role swc:heldBy ?author .
        ?proceedings swc:relatedToEvent eswc:2010.
        ?paper swrc:author ?author .
        ?author foaf:based_near ?place.
        ?paper swc:isPartOf ?proceedings .
        }
optional
{
        SERVICE <dbpedia-subset endpoint>
        {
        ?place dbpedia:capital ?capital;
        geo:lat ?latitude;
        geo:long ?longititude.
        }
}
}
```

LargeRDFBench queries.

```
############################ C8: SPARQL 1.0 ############################
#For all the proceedings of ISWC along with proceeding address and the list of
    authors, their complete name, their corresponding publications,
their affiliations and in which country they actually based in with the optional
    information of capital, language, government type, leader name and population
    density of their country.
SELECT DISTINCT *
WHERE
{
?paper swc:isPartOf iswc:proceedings .
iswc:proceedings swrc:address ?proceedingAddress.
?paper swrc:author ?author .
?author swrc:affiliation ?affiliation ;
 rdfs:label ?fullnames ;
 foaf:based_near ?place.
OPTIONAL
{
?place dbpedia:capital ?capital;
       dbpedia:populationDensity  ?populationDensity;
       dbpedia:governmentType   ?governmentType;
       dbpedia:language  ?language ;
       dbpedia:leaderTitle ?leaderTitle.
}
}
#---------------------- C8: SPARQL 1.1 ----------------------
SELECT DISTINCT * WHERE
{
        SERVICE <swdf endpoint>
        {
        ?paper swc:isPartOf iswc:proceedings .
        iswc:proceedings swrc:address ?proceedingAddress.
        ?paper swrc:author ?author .
        ?author swrc:affiliation ?affiliation ;
        rdfs:label ?fullnames ;
        foaf:based_near ?place.
        }
optional
{
        SERVICE <dbpedia-subset endpoint>
        {
        ?place dbpedia:capital ?capital;
         dbpedia:populationDensity  ?populationDensity;
        dbpedia:governmentType   ?governmentType;
        dbpedia:language  ?language ;
        dbpedia:leaderTitle ?leaderTitle.
        }
}
}
```

LargeRDFBench queries.

```
############################ C9: SPARQL 1.0 ############################
#For all drugs in DBpedia, find all drugs they interact with each other, along with
    the details of the interaction, with optional information of drug description,
    its structure and casRegistry Number for one of the interacted drug that affects
     hummans and mammals.
SELECT *
WHERE
{
?Drug rdf:type dbpedia:Drug .
?drugbankDrug owl:sameAs ?Drug .
?InteractionName drugbank:interactionDrug1 ?drugbankDrug .
?InteractionName drugbank:interactionDrug2 ?drugbankDrug2 .
?InteractionName drugbank:text ?IntEffect
OPTIONAL
{
?drugbankDrug  drugbank:affectedOrganism 'Humans_and_other_mammals';
drugbank:description ?description ;
drugbank:structure ?structure ;
drugbank:casRegistryNumber ?casRegistryNumber
}
}
ORDER BY (?drugbankDrug)
LIMIT 100
#---------------------- C9: SPARQL 1.1 ----------------------
SELECT * WHERE
{
        SERVICE <dbpedia-subset endpoint> { ?Drug rdf:type dbpedia:Drug .}
        SERVICE <drugbank endpoint>
        {
        ?drugbankDrug owl:sameAs ?Drug .
        ?InteractionName drugbank:interactionDrug1 ?drugbankDrug .
        ?InteractionName drugbank:interactionDrug2 ?drugbankDrug2 .
        ?InteractionName drugbank:text ?IntEffect
        }
optional
{
        SERVICE <drugbank endpoint>
        {
        ?drugbankDrug  drugbank:affectedOrganism 'Humans_and_other_mammals';
        drugbank:description ?description ;
        drugbank:structure ?structure ;
        drugbank:casRegistryNumber ?casRegistryNumber
        }
}
}
order by (?drugbankDrug)
limit 100
```

LargeRDFBench queries.

```
############################ C10: SPARQL 1.0 ############################
#Get clinical information about TCGA patient along with drug and location
    information.
SELECT DISTINCT ?patient  ?gender ?country ?popDensity ?drugName ?indication ?
    formula ?compound
WHERE
{
?uri tcga:bcr_patient_barcode                    ?patient .
?patient tcga:gender                  ?gender .
?patient dbpedia:country              ?country .
?country dbpedia:populationDensity               ?popDensity .
?patient tcga:bcr_drug_barcode                   ?drugbcr .
?drugbcr tcga:drug_name               ?drugName .
?drgBnkDrg  drugbank:genericName                 ?drugName .
?drgBnkDrg  drugbank:indication                  ?indication .
?drgBnkDrg  drugbank:chemicalFormula  ?formula .
?drgBnkDrg      drugbank:keggCompoundId          ?compound .
}
#---------------------- C10: SPARQL 1.1 ----------------------
SELECT DISTINCT ?patient  ?gender ?country ?popDensity ?drugName ?indication ?
    formula ?compound ?ChemicalEquation ?ReactionTitle
WHERE
{
        SERVICE <LinkedTCGA-A endpoint>
        {
        ?uri tcga:bcr_patient_barcode                    ?patient .
        ?patient tcga:gender                                     ?gender.
        ?patient dbpedia:country                                 ?country.
        ?patient tcga:bcr_drug_barcode                   ?drugbcr.
        ?drugbcr tcga:drug_name                                  ?drugName.
        }
        SERVICE <dbpedia-subset endpoint> {?country dbpedia:populationDensity
                    ?popDensity.}
        SERVICE <drugbank endpoint>
        {
        ?drgBnkDrg  drugbank:genericName                 ?drugName.
        ?drgBnkDrg  drugbank:indication                  ?indication.
        ?drgBnkDrg  drugbank:chemicalFormula  ?formula.
        ?drgBnkDrg      drugbank:keggCompoundId          ?compound .
        }
}
```

LargeRDFBench queries.

```
############################ L1: SPARQL 1.0 ############################
#Get the exon and gene expression values for TCGA patient no. TCGA-37-3789.
SELECT  ?expValue
WHERE
{
  {
   ?s   tcga:bcr_patient_barcode        <http://tcga.deri.ie/TCGA-37-3789>.
   <http://tcga.deri.ie/TCGA-37-3789>   tcga:result     ?results.
   ?results  tcga:RPKM ?expValue.
  }
UNION
  {
   ?uri tcga:bcr_patient_barcode        <http://tcga.deri.ie/TCGA-37-3789>.
   <http://tcga.deri.ie/TCGA-37-3789>   tcga:result     ?geneResults.
   ?geneResults  tcga:scaled_estimate ?expValue.
  }
}
#---------------------- L1: SPARQL 1.1 ----------------------
SELECT   ?expValue
WHERE
{
  {
   SERVICE <LinkedTCGA-E endpoint>
   {
   ?s   tcga:bcr_patient_barcode        <http://tcga.deri.ie/TCGA-37-3789>.
   <http://tcga.deri.ie/TCGA-37-3789>   tcga:result     ?results.
   ?results  tcga:RPKM ?expValue.
   }
  }
UNION
  {
   SERVICE <LinkedTCGA-A endpoint>
   {
   ?uri tcga:bcr_patient_barcode        <http://tcga.deri.ie/TCGA-37-3789>.
   <http://tcga.deri.ie/TCGA-37-3789>   tcga:result     ?geneResults.
   ?geneResults  tcga:scaled_estimate ?expValue.
   }
  }
}
############################ L2: SPARQL 1.0 ############################
# Get the tumor type and Exon values for all the patient having lung cancer and
    tumor weight less than 56.
SELECT   DISTINCT ?patient ?tumorType ?exonValue
WHERE
{
?s tcga:bcr_patient_barcode ?patient.
?patient tcga:disease_acronym <http://tcga.deri.ie/lusc>.
?patient tcga:tumor_weight ?weight.
?patient tcga:tumor_type ?tumorType.
?patient tcga:result ?results.
?results tcga:RPKM  ?exonValue.
FILTER(?weight <= 55)
}
```

```
#----------------------- L2: SPARQL 1.1 -----------------------
SELECT   DISTINCT ?patient ?tumorType ?exonValue
WHERE
{
SERVICE <LinkedTCGA-A endpoint>
   {
        ?s tcga:bcr_patient_barcode ?patient.
        ?patient tcga:disease_acronym <http://tcga.deri.ie/lusc>.
        ?patient tcga:tumor_weight ?weight.
        ?patient tcga:tumor_type ?tumorType.
        }
        SERVICE <LinkedTCGA-E endpoint>
        {
        ?patient tcga:result ?results.
        ?results tcga:RPKM  ?exonValue.
        }
FILTER(?weight <= 55)
}
############################ L3: SPARQL 1.0 ############################
# Get the methylation values for all the patients who have been treated drug "
    Tarceva" and they died while at initial pathologic diagnosis age of less than
    52.
WHERE
{
?s tcga:bcr_patient_barcode ?patient.
?patient tcga:vital_status "Dead".
?patient tcga:bcr_drug_barcode ?drug.
?drug tcga:drug_name "Tarceva".
?patient tcga:age_at_initial_pathologic_diagnosis ?age.
?patient tcga:result ?results.
?results  tcga:beta_value ?methylationValue.
FILTER(?age <= 51)
}
ORDER BY (?patient)
#----------------------- L3: SPARQL 1.1 -----------------------
SELECT  ?patient ?methylationValue
WHERE
{
        SERVICE <LinkedTCGA-A endpoint>
   {
        ?s tcga:bcr_patient_barcode ?patient.
        ?patient <http://tcga.deri.ie/schema/vital_status> "Dead".
        ?patient tcga:bcr_drug_barcode ?drug.
        ?drug tcga:drug_name "Tarceva".
        ?patient <http://tcga.deri.ie/schema/age_at_initial_pathologic_diagnosis> ?
            age.
        }
        SERVICE <LinkedTCGA-M endpoint>
   {
        ?patient tcga:result ?results.
        ?results  tcga:beta_value ?methylationValue.
        }
FILTER(?age <= 51)
}
ORDER BY (?patient)
```

LargeRDFBench queries.

```
############################ L4: SPARQL 1.0 ############################
#Get the expression values for all the patients either belong to Brazil or Argentina
    . SELECT ?expressionValues
WHERE
{
 {
  ?uri tcga:bcr_patient_barcode ?patient.
  ?patient dbpedia:country ?country.
  ?patient tcga:result ?results.
  ?results  tcga:reads_per_million_miRNA_mapped ?expressionValues.
 }
UNION
 {
  ?s tcga:bcr_patient_barcode ?patient.
  ?patient dbpedia:country ?country.
  ?patient tcga:result ?exonResults.
  ?exonResults tcga:RPKM ?expressionValues.
 }
FILTER REGEX(?country,"Brazil|Argentina", "i")
}
#---------------------- L4: SPARQL 1.1 ----------------------
SELECT    ?expressionValues
WHERE
{
 {
  SERVICE <LinkedTCGA-A endpoint>
   {
  ?uri tcga:bcr_patient_barcode ?patient.
  ?patient dbpedia:country ?country.
  ?patient tcga:result ?results.
  ?results  tcga:reads_per_million_miRNA_mapped ?expressionValues.
   FILTER REGEX(?country,"Brazil|Argentina", "i")
  }
 }
UNION
 {
 SERVICE <LinkedTCGA-A endpoint>
  {
  ?patient dbpedia:country ?country.
  ?s tcga:bcr_patient_barcode ?patient.
  FILTER REGEX(?country,"Brazil|Argentina", "i")
  }
  SERVICE <LinkedTCGA-E endpoint>
   {
  ?patient tcga:result ?exonResults.
  ?exonResults tcga:RPKM ?expressionValues.
  }
 }

}
```

LargeRDFBench queries.

```
############################ L5: SPARQL 1.0 ############################
#Get the methylation values for CNTNAP2 gene of all the cancer patients.
SELECT  ?methylationCNTNAP2
WHERE
{
 ?s affymetrix:x-symbol <http://bio2rdf.org/symbol:CNTNAP2>.
 ?s affymetrix:x-geneid ?geneId.
 ?geneId rdf:type tcga:expression_gene_lookup.
 ?geneId tcga:chromosome ?lookupChromosome.
 ?geneId tcga:start ?start.
 ?geneId tcga:stop  ?stop.
 ?uri tcga:bcr_patient_barcode ?patient .
 ?patient tcga:result ?recordNo .
 ?recordNo tcga:chromosome   ?chromosome.
 ?recordNo tcga:position     ?position.
 ?recordNo tcga:beta_value  ?methylationCNTNAP2.
 FILTER (?position >= ?start && ?position <= ?stop && str(?chromosome) = str(?
     lookupChromosome) )
 }
#---------------------- L5: SPARQL 1.1 ----------------------
SELECT  ?methylationCNTNAP2
WHERE
{
        SERVICE <affymetrix endpoint>
   {
        ?s affymetrix:x-symbol <http://bio2rdf.org/symbol:CNTNAP2>.
        ?s affymetrix:x-geneid ?geneId.
        }
        SERVICE <LinkedTCGA-A endpoint>
   {
        ?geneId rdf:type tcga:expression_gene_lookup.
        ?geneId tcga:chromosome ?lookupChromosome.
        ?geneId tcga:start ?start.
        ?geneId tcga:stop  ?stop.
        }
        SERVICE <LinkedTCGA-M endpoint>
   {
        ?uri tcga:bcr_patient_barcode ?patient .
        ?patient tcga:result ?recordNo .
        ?recordNo tcga:chromosome   ?chromosome.
        ?recordNo tcga:position     ?position.
        ?recordNo tcga:beta_value  ?methylationCNTNAP2.
        }
 FILTER (?position >= ?start && ?position <= ?stop && str(?chromosome) = str(?
     lookupChromosome) )
}
```

LargeRDFBench queries.

```
############################ L6: SPARQL 1.0 ############################
#Description: For all cancer patients, get the genomic locations and corresponding
    gene expression values for chromosome associated with KRAS gene.
SELECT DISTINCT ?patient ?start ?stop ?geneExpVal
WHERE
{
        ?s affymetrix:x-symbol <http://bio2rdf.org/symbol:KRAS>.
        ?s affymetrix:x-geneid ?geneId.
        ?geneId rdf:type tcga:expression_gene_lookup.
        ?geneId tcga:chromosome ?lookupChromosome.
        ?uri tcga:bcr_patient_barcode ?patient .
        ?patient tcga:result ?recordNo .
        ?recordNo tcga:chromosome   ?chromosome.
        ?recordNo tcga:start ?start.
        ?recordNo tcga:stop ?stop.
        ?recordNo tcga:scaled_estimate ?geneExpVal
        FILTER (str(?lookupChromosome)= str(?chromosome))
}
#---------------------- L6: SPARQL 1.1 ----------------------
SELECT DISTINCT ?patient ?start ?stop ?geneExpVal
WHERE
{
        SERVICE <affymetrix endpoint>
   {
        ?s affymetrix:x-symbol <http://bio2rdf.org/symbol:KRAS>.
        ?s affymetrix:x-geneid ?geneId.
        }
 SERVICE <LinkedTCGA-A endpoint>
   {
        ?geneId rdf:type tcga:expression_gene_lookup.
        ?geneId tcga:chromosome ?lookupChromosome.
        ?uri tcga:bcr_patient_barcode ?patient .
        ?patient tcga:result ?recordNo .
        ?recordNo tcga:chromosome   ?chromosome.
        ?recordNo tcga:start ?start.
        ?recordNo tcga:stop ?stop.
        ?recordNo tcga:scaled_estimate ?geneExpVal
        }
                FILTER (str(?lookupChromosome)= str(?chromosome))
}
```

LargeRDFBench queries.

```
############################ L7: SPARQL 1.0 ############################
#Find the clinical aliquot data for all the patients belonging to those countries
    with population density greater than 31.
SELECT DISTINCT ?patient ?p ?o
WHERE
{
?uri tcga:bcr_patient_barcode ?patient .
?patient dbpedia:country ?country.
?country dbpedia:populationDensity ?popDensity.
?patient tcga:bcr_aliquot_barcode ?aliquot.
?aliquot ?p ?o.
FILTER(?popDensity >= 32)
}
#---------------------- L7: SPARQL 1.1 ----------------------
SELECT DISTINCT ?patient ?p ?o
WHERE
{
        SERVICE <LinkedTCGA-A endpoint>
    {
        ?uri tcga:bcr_patient_barcode ?patient .
        ?patient dbpedia:country ?country.
        }
        SERVICE <http://dbpedia-subset endpoint> { ?country dbpedia:
            populationDensity ?popDensity. }
        SERVICE <LinkedTCGA-A endpoint>
    {
        ?patient tcga:bcr_aliquot_barcode ?aliquot.
        ?aliquot ?p ?o.
     }
FILTER(?popDensity >= 32)
}
############################ L8: SPARQL 1.0 ############################
#Get the outliers expression values for TCGA patient TCGA-D9-A1X3.
SELECT  ?chromosome ?expressionValue
WHERE
{
{
?uri tcga:bcr_patient_barcode <http://tcga.deri.ie/TCGA-D9-A1X3> .
<http://tcga.deri.ie/TCGA-D9-A1X3> tcga:result ?recordNo .
?recordNo tcga:chromosome   ?chromosome.
?recordNo tcga:protein_expression_value ?expressionValue.
}
UNION
{
?s tcga:bcr_patient_barcode <http://tcga.deri.ie/TCGA-D9-A1X3> .
<http://tcga.deri.ie/TCGA-D9-A1X3> tcga:result ?results .
?results tcga:chromosome   ?chromosome.
?results tcga:beta_value ?expressionValue.
}
FILTER (?expressionValue > 0.05)
}
```

LargeRDFBench queries.

```
#----------------------- L8: SPARQL 1.1 -----------------------
SELECT  ?chromosome ?expressionValue
WHERE
{
{
        SERVICE <LinkedTCGA-A endpoint>
    {
        ?uri tcga:bcr_patient_barcode <http://tcga.deri.ie/TCGA-D9-A1X3> .
        <http://tcga.deri.ie/TCGA-D9-A1X3> tcga:result ?recordNo .
        ?recordNo tcga:chromosome   ?chromosome.
        ?recordNo tcga:protein_expression_value ?expressionValue.
        }
}
UNION
{
        SERVICE <LinkedTCGA-M endpoint>
    {
        ?s tcga:bcr_patient_barcode <http://tcga.deri.ie/TCGA-D9-A1X3> .
        <http://tcga.deri.ie/TCGA-D9-A1X3> tcga:result ?results .
        ?results tcga:chromosome   ?chromosome.
        ?results tcga:beta_value ?expressionValue.
        }
}
FILTER (?expressionValue > 0.05)
}
```