LargeRDFBench: A Billion Triples Benchmark for SPARQL Endpoint Federation

Muhammad Saleem[1], Ali Hasnain[2], and Axel-Cyrille Ngonga Ngomo[1]

[1] Universität Leipzig, IFI/AKSW, PO 100920, D-04009 Leipzig
{lastname}@informatik.uni-leipzig.de
[2] Insight Center for Data Analytics, National University of Ireland, Galway
ali.hasnain@insight-centre.org

Abstract. The Web of Data is now a compendium of thousands of interlinked datasets containing billions of facts pertaining to millions of resources. Gathering information from these large and distributed data sources 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 current benchmarks are either synthetic or too small in size and complexity. Furthermore, state-of-the-art federated SPARQL benchmarks mostly focused on a single performance criterion, i.e., the overall query runtime. Thus, they cannot provide a more fine-grained evaluation of the systems. 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, result completeness and correctness. Our evaluation results suggest that the performance of current SPARQL query federation systems on simple queries 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.

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 datasets is required to satisfy the needs of the user. The importance of Linked Data management and SPARQL queries has led to the development of several benchmarks (e.g., [1,3,7,10,16,17,20]) that allow assessing the performance of SPARQL query processing systems. However, all of these benchmarks (except FedBench [16]) have focused on the problem of query evaluation over local, centralised repositories. Hence, these benchmarks do not consider federated queries over multiple interlinked datasets hosted by different SPARQL endpoints. Moreover, many of them either rely on synthetic data (e.g., [3,7,17]) or synthetic queries [1].

While synthetic benchmarks allow generating datasets of virtually any size, they often fail to reflect reality [4]. In particular, previous works [4] point out that artificial benchmarks are typically highly structured while real Linked Data sources are less structured. Moreover, the synthetic queries should reflect the characteristics of the real queries (i.e., they should show typical requests on the underlying datasets) [2,11]. Thus, synthetic benchmark results are rarely sufficient to extrapolate the performance of federation engines when faced with real data and in real queries. A trend towards benchmarks with real data and real queries (e.g., FedBench [16], DBPSB [10], BioBenchmark [20]) has thus been pursued over the last 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. While the trend towards using real data and real queries in benchmarks addresses the need to reflect reality, most of the current benchmarks for SPARQL query execution mostly focused on a single performance criterion, i.e., the query execution time. Thus, they fail to provide results that allow a more fine-grained evaluation of SPARQL query processing systems to detect the components of systems that need to be improved [9,13]. For example, performance metrics such as the *completeness and correctness of result sets* and the *effectiveness of source selection* both in terms of *total number of data sources selected*, and the corresponding *source selection time* are not addressed in the existing benchmarks [9,13].

In this paper, we address these drawbacks by presenting LargeRDFBench. (1) 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. (2) We provide the SPARQL 1.0 and SPARQL 1.1 versions of the queries that are mostly collected from domain experts. In particular, we provide three types of queries that allow evaluating different aspects of the scalability of the current query federation frameworks. Note that for a particular LargeRDFBench query, the SPARQL 1.0 and SPARQL 1.1 versions of the query retrieve exactly the same result set. (3) 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, 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. (4) Furthermore, we show that the current ranking of these systems based on simple queries differs significantly from their ranking when on more complex queries.

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). Thereafter, we point out the current drawbacks of existing benchmarks in more detail (Section 3). In Section 4, we describe LargeRDFBench. An evaluation of state-of-the-art systems based on LargeRDFBench follows next. The results are discussed and we finally conclude. The benchmark and further evaluation results can be found at benchmark homepage.[3]

2 Background

This section explains the main components of the 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, (B) a set of SPARQL queries and (3) a set of performance metrics.

[3] LargeRDFBench `http://bigrdfbench.googlecode.com`

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

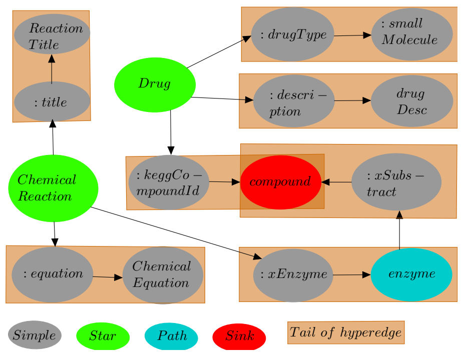


Fig. 1: DH representation of the SPARQL query. Prefixes are ignored for simplicity

Datasets: The *datasets* used in the federated SPARQL benchmark should complement each other 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, outdegrees and their resource wise distributions [4]. Duan et al. [4] combines all of these datasets features into a single composite metric called *structuredness or cohesion*. For a given dataset, the structuredness value ranges [0,1] with 0 means less structured and 1 means high structured dataset. A federated SPARQL query benchmark should comprise datasets of varying structuredness values.

SPARQL Queries: We represent each basic graph patterns (BGP) of a SPARQL query as a directed hypergraph (DH) according to [13]. We chose this representation because it allows representing property-property joins, which previous works [1,6] do not allow to model. The DH representation of a BGP is formally defined as follows:

Definition 1. *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;*
- *λ_{vt} is a vertex-type-assignment function. Given a vertex $v \in V$, its vertex type can be '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 the complete SPARQL query as DH is the union of the representations of its BGPs. As an example, the DH representation of the query is shown in Figure 1. Based on the DH representation of SPARQL queries we can define:

Definition 2 (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 set of vertices of type 'sink' in a DH representation of a SPARQL query, then the total number of join vertices in the query $\#JV$ = $|ST| + |PT| + |HB| + |SN|$.*

Definition 3 (Join Vertex Degree). *The DH representation of the 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 denoted as JVD_v = $|E_{in}(v)| + |E_{out}(v)|$.*

Definition 4 (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.*[4] *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 5 (Total Triple Pattern-wise Sources). *By using Definition 4, we can define the total triple pattern-wise sources selected for query Q as the sum of the magnitudes of relevant sources set R_i over all individual triple patterns $tp_i \in Q$.*

Definition 6 (Number of Sources Span). *The number of sources that potentially contribute to the query result set (sources span for short) are those that are* relevant *to at least one triple pattern in the query [18]. However, since triple patterns with common query predicates such as `rdf:type` and `owl:sameAs` are likely to be found in all data sources (i.e., all sources are* relevant*), we only count a source if it is also* relevant *to at least one more triple pattern in the same query [13].*

Definition 7 (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 N_m be the total number of triples in d that matches tp_i, then the selectivity of tp_i w.r.t d denoted by $Sel(tp_i, d) = N_m/N$. The selectivity of the filtered triple pattern (a triple pattern with SPARQL `FILTER` clause) is calculated in the same way.*

According to previous works [1,6], 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), 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 [9,13] 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*, and the *overall*

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

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 be considered for live SPARQL endpoints query processing. However, it is almost negligible for local, dedicated network setup used in our evaluation. Note that LargeRDFBench is a benchmark for SPARQL endpoint federation, in contrast to Linked Data federation [13].

3 Related work

A large 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) [3], the Lehigh University Benchmark (LUBM) [7], the DBpedia Sparql Benchmark (DBPSB) [10], FedBench [16], SP^{2}Bench [17], and the BioBenchmark [20]. WSTB, BSBM, DBPSB, SP^{2}Bench, 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 [6] is a heuristic for automatic generation of federated SPARQL queries which is limited to conjunctive BGPs. Non-conjunctive queries that make use of the SPARQL `UNION`, `OPTIONAL` clauses are not considered. Thus, the generated set of synthetic queries fails to reflect the characteristics of the real queries. For example, the DBpedia query log [11] shows that 20.87%, 30.02% of the real queries contains SPARQL `UNION` and `FILTER` clauses, respectively. However, both of these clauses are not considered in SPLODGE queries generation. Moreover, the use of different SPARQL clauses and triple pattern join types greatly varies from one dataset to another dataset, thus making it almost impossible for automatic query generator to reflect the reality. For example, the DBpedia and Semantic Web Dog Food (SWDF) query log [2] shows that the use of the SPARQL `LIMIT` (27.99% for SWDF vs 1.04% for DBpedia) and `OPTIONAL` (0.41% for SWDF vs 16.61% for DBpedia) clauses greatly varies for these two datasets.

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 [18,5,8,13,14]. However, the real queries (excluding synthetic SP^{2}Bench benchmark queries) are low in complexity. 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).

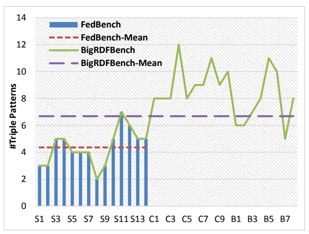
Table 1: Queries distribution with respect to SPARQL clauses and join vertex types.

	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%

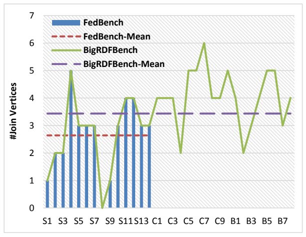
As mentioned before, we are only interested in SPARQL endpoint federation queries. Therefore, the 14 SPARQL endpoint federation queries are further discussed in rest of the paper.

Table 1 and Figure 2 show that the FedBench SPARQL endpoint federation queries are also low in complexity and do not sufficiently complement (in terms of standard deviations for various query features discussed in previous section) each other's. Consequently, they may favour (in fact our evaluation given in Section 5.2 shows that indeed this is the case) a particular type of federation system. 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 on the lower side. 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 2e) 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 2f). The important SPARQL clauses such `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). Most importantly, the average query execution is small (about 2 seconds on average ref. Section 5.2). Finally, FedBench rely 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. Furthermore, a more fine-grained evaluation of the federation engines, to detect the components that need to be improved is not possible [9].

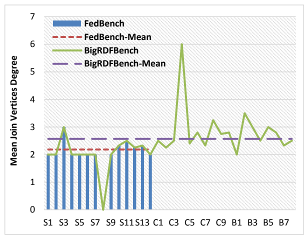
To address these limitations, we propose LargeRDFBench, a billion-triple benchmark which encompasses a total of 13 real, interconnected datasets of varying *structuredness* (ref. Figure 3a) and real queries of varying complexities. (see Table 1 and Figure 2). Our benchmark includes all of the 14 SPARQL 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 2e) and intermediary results (see triple pattern selectivities, Figure 2f) . 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 is able to fully answer the real use-case large data queries.



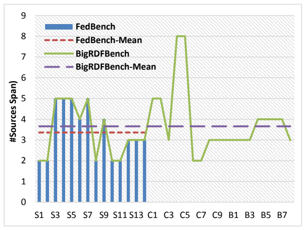
(a) #TP. (±1.33, ±2.64)



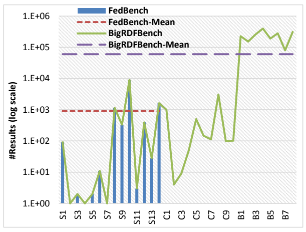
(b) #JV. (±1.33, ±1.41)



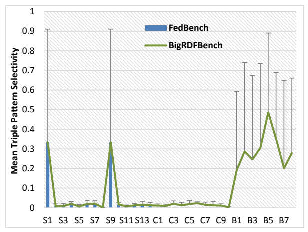
(c) MJVD. (±0.30, ±0.76)



(d) #SS. (±1.27, ±1.53)



(e) #R. (±2397, ±113763)



(f) MTPS. Mean (±0.11, ±0.14)

Fig. 2: Comparison of query characteristics of FedBench and LargeRDFBench. **#TP** = Number of triple patterns, **#JV** = Number of join vertices, **MJVD** = Mean join vertices degree, **#SS** = Number of sources span, **#R** = Number of results, **MTPS** = Mean Triple Pattern Selectivity, Values inside bracket shows standard deviations for FedBench and LargeRDFBench, respectively. X-axis shows the query name.

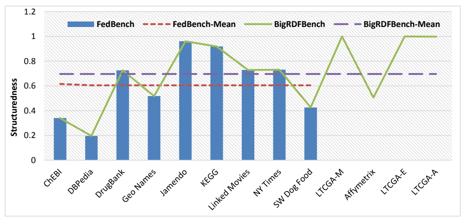
4 Benchmark Description

The idea behind this work was to design a benchmark based on real data and real queries that implements all of the key benchmark components 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 3b). Furthermore, some of the datasets are highly *structured* while others are low *structured* (ref. Figure 3a). 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 degress, 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) real queries mostly collected from domain experts and 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.

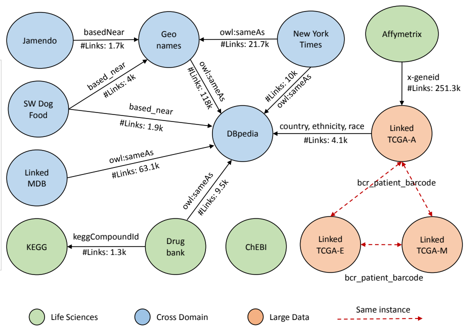
4.1 Benchmark Datasets

Our benchmark consists of a total of 13 real-world datasets[5] of which 12 are interlinked. The datasets were collected from different domains as shown in Figure 3b. We began by selecting all nine real-world datasets from Fedbench [16]. We added three sub-datasets

[5] Live SPARQL endpoints, datadumps URL's are given at project homepage: See footnote 3



(a) Structuredness (±0.26, ±0.28)



(b) Connectivity

Fig. 3: LargeRDFBench datasets structuredness and connectify

from three different Linked TCGA live SPARQL endpoints [15] (i.e. Linked TCGA-A, Linked TCGA-M, and Linked TCGA-E) along with Affymetrix. We chose Linked TCGA because it is one of the first datasets that abides by many of the Vs of large data (Volume, Velocity, Value, ...) [12]. 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 2e and benchmark homepage) into the benchmark. Figure 3b shows the topology of all 13 datasets in LargeRDFBench while some 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 on project homepage. It is important to 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 subsection Large Data). The datasets in LargeRDFBench belong to three categories: Cross-domain, Life Sciences domain and large data.

Cross-domain Datasets This category comprises datasets which pertain to several domains including news, movies, music, semantic web conferences and geography. These datasets include a 1) subset from DBpedia comprising infoboxes and the instance types, 2) the music knowledge base called Jamendo, 3) LinkedMDB , a knowledge base containing movie and actor information, 4) GeoNames , which contains geographical data about persons, locations, as well as organisations, 5) Semantic Web Dog Food which describes Semantic Web conferences and publications, and 6) a knowledge base containing news from the New York Times . Figure 3b shows the links between these data sources.

Life Sciences Domain Life Sciences domain data sources include 1) Drugbank, a knowledge base containing information pertaining to drugs, their composition and their interactions with other drugs, 2) the Kyoto Encyclopedia of Genes and Genomes (KEGG) which contains further information about chemical compounds and reactions with a focus on information relevant for geneticists, 3) the Chemical Entities of Biological Interest (ChEBI) knowledge base which describes the life sciences domain from a chemical point of view and 4) the Affymetrix dataset that contains the probesets used in the Affymetrix microarrays.

Large Data: Linked TCGA Linked TCGA is the RDF version of Cancer Genome Atlas[6] (TCGA) presented in [15]. 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). Currently, Linked TCGA comprises a total of 20.4 billion triples[7] 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.

We selected the data of 306 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. Similarly, we created three different datasets namely 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 C1-C10 and the 8 large dubbed L1-L8 were created by the authors with the help of domain experts. Table 1 and Figure 2 shows key features of these queries. Further query statistics can be found on project home page.

Simple Queries In comparison to the other queries in the benchmark, the queries in this category comprise the smallest number of triple patterns, which ranges from 2 to 7. These queries require retrieving data from 2 to 5 data sources (ref. Figure 2d). 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 are small (around 2 seconds for FedX).

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 SPARQL clauses, and the small query execution times. Consequently, queries in this category

[6] `http://cancergenome.nih.gov/`
[7] `http://tcga.deri.ie/`

rely on at least 8 triple patterns. 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 reaches up to more than 10 minutes.

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 data sets (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 2f) or lead to large result sets (minimum 80459, see Figure 2e). 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 [9,13] suggest that the following five metrics are important to evaluate the performance of federation engines: (1) the total number of triple pattern-wise (TPW) source 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 and (5) the average query execution time. The source index/data summaries generation time and index compression ratio (i.e., index to dataset ratio) are also of central importance when evaluating endpoints. However, they are not applicable to index-free approaches such as FedX [18]. Previous works [13] 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 turns added into the overall query execution time.

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 endpoint used in our experiments was installed on a separate machine. The specification of each of the machines is given on the project home page. 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. All experiments (i.e., the federation engines themselves) were ran 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. The experiments were carried out in a local network, so the network costs were negligible. 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 1 hour (3.6×10^6 ms) for large data queries. Furthermore, the query runtime results were analyzed statistically using Wilcoxon Signed Rank Test (WSRT). We chose this test because it is parameter-free and does not assume a particular error distribution in the data like a t-test does. For all the significance tests, we set the p-value to 0.05.

We compared five SPARQL endpoint federation engines (versions available as of October 2014) – FedX [18], SPLENDID [5], ANAPSID [5], FedX+HiBISCuS [13], SPLENDID+HiBISCuS [13] – on all of the 32 benchmark queries. Note that HiBISCuS [13] 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 [13]. Of all the systems, only ANAPSID and HiBISCuS perform *join-aware* Triple Pattern-Wise Source Selection (TPWSS).

5.2 SPARQL 1.0 Experimental Results

Efficiency of Source Selection We define efficient source selection in terms of: (1) the total number of triple pattern-wise sources selected (#TP), (2) the total number of SPARQL ASK requests (#AR) used to obtain (1), and (3) the source selection time (SST). Table 2 shows the results of these three metrics for the selected approaches.

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 2). 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 (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 source 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 of the approach being based on making use of different URI authorities to perform source pruning [13]. 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) [8] to skip a large number of sources. However, SPARQL ASK queries are expensive compare to local index lookups, as performed in HiBISCuS.

Table 2: Comparison of the source selection for simple, complex, and large data queries in terms of total triple pattern-wise sources selected **#TP**, total number of SPARQL ASK requests **#AR**, and average source selection time **SST** in msec. **SST*** represents the average source selection time for FedX (100% cached i.e. #A =0 for all queries). Complete table for individual queries is provided in the extended version of the paper given at `http://goo.gl/XPHlzq`

	FedX				SPLENDID			ANAPSID			HiBISCuS			Optimal
Query Ctg.	#TP	#AR	SST	SST*	#TP	#AR	SST	#TP	#AR	STT	#TP	#AR	SST	#TP
Simple	142	793	258	**5**	142	93	355	80	114	165	**78**	65	**150**	**67**
Complex	199	1196	363	**6**	199	11	310	111	103	253	**106**	**0**	**162**	**101**
Large Data	133	793	337	**6**	133	44	372	**64**	135	3317	118	**0**	**140**	**61**
Net	474	2782	311	**6**	474	148	345	**255**	352	980	302	**65**	**151**	**229**

Table 3: Result set completeness and correctness: Systems with incomplete precision and recall. The values inside brackets show the LargeRDFBench query version, i.e., SPARQL 1.0 and SPARQL 1.1. For queries L2, L3, and L5 FedX and its HiBiSCUS extension produced zero results, thus both precision, recall is zero and F1 is undefined for these queries. (**NA** = SPARQL 1.1 not applicable, **TO** = Time out)

System	C7 (v1.0, v1.1)			L1 (v1.0, v1.1)			S14 (v1.1)			C6 (v1.1)		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
FedX	0.25	0.19	0.22	TO	TO	TO	1	0.65	0.78	1	0.97	0.98
FedX+HiBISCuS	0.25	0.19	0.22	TO	TO	TO	NA	NA	NA	NA	NA	NA
ANAPSID	1	1	1	1	0.14	0.25	1	1	1	1	1	1

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 3 shows the federation engines and the corresponding LargeRDFBench queries for which complete and correct results were not retrieved by at least one of the system. All those queries for which every system either timed out or resulted in runtime errors are not included, since results completeness and correctness cannot be determined in such cases. Here, SPLENDID and its HiBISCuS extension are the only systems that provide complete and correct results provided that the query is fully executed within the time out limit. The incomplete results generated by some of the systems can be due to a number of reasons, e.g., their join implementation, the type of network [9], the use of an outdated index or cache or even endpoints restrictions on the maximum result set sizes. However, in our evaluation we always used an up-to-date index and cache, there was no 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 type of the *join* used and its implementation. For query C7, FedX and its HiBISCuS extension were able to retrieve 85 records instead of the actual result set size, i.e., 112. For query L1, ANAPSID retrieved 35810 results while the actual result set size for this query is 227192. For the same query, both SPLENDID and FedX along with their extensions were only able to produce less than 30K results within timeout limit of one hour. For the SPARQL 1.1 versions of S14, FedX retrieved 1054 of the expected 1620 results while for C6 145 of the 148 results were retrieved.

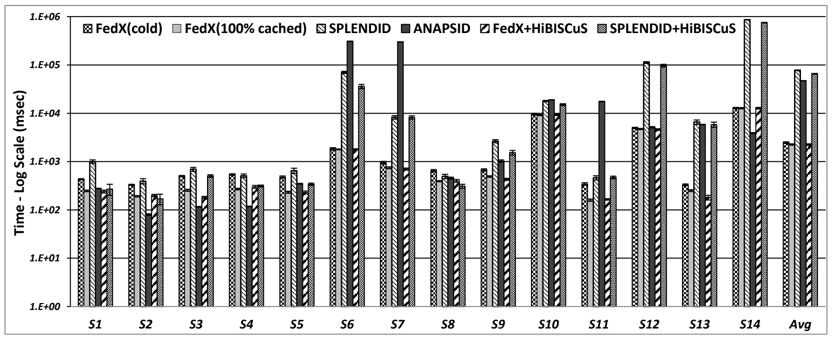


Fig. 4: Query execution time for simple category queries.

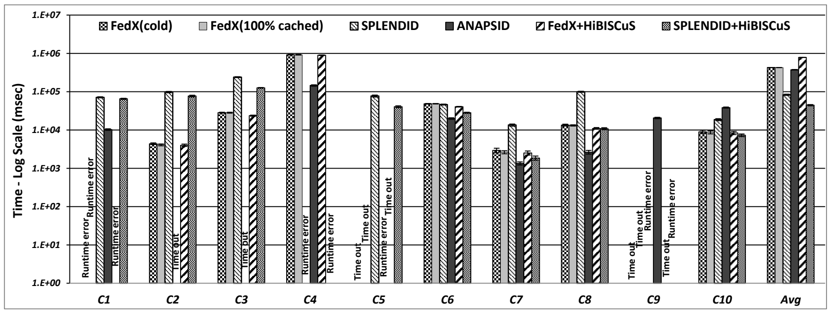


Fig. 5: Query execution time for complex category queries.

Query Execution Time The query execution time has often been used as key metric to compare federation engines. Figure 4, Figure 5, and Table 4 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 4 and Figure 5. 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.

- *Simple queries*: FedX+HiBISCuS and FedX clearly outperform the remaining systems (see Figure 4). 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).
- *Complex queries*: SPLENDID+HiBISCuS performed better than SPLENDID and was followed by ANAPSID, FedX+HiBISCuS and FedX. ANAPSID is better than SPLENDID+HiBISCuS in 4/7 comparable (those for which complete and correct results are retrieved by both systems) queries (all significant improvements on WSRT), SPLENDID+HiBISCuS is better than FedX and FedX+HiBISCuS in 5/7

Table 4: Runtimes on large data queries. **F(c)** = FedX (cold), **F(w)** = FedX(100% cached), **S** = SPLENDID, **A** = ANAPSID, **F+H** = FedX+HiBISCuS, **S+H** = SPLENDID+HiBISCuS. (**TO** = Time out after 1 hour, **ZR** = zero results, **IR** = incomplete results, **RE** = runtime error). Times are in seconds.

Query	F(c)	F(w)	S	A	F+H	S+H	Query	F(c)	F(w)	S	A	F+H	S+H
L1	TO	TO	TO	IR	TO	TO	L5	ZR	ZR	TO	RE	ZR	TO
L2	ZR	ZR	TO	TO	ZR	TO	L6	TO	TO	TO	TO	TO	TO
L3	ZR	ZR	TO	ZR	ZR	TO	L7	122	122	114	105	119	114
L4	TO	TO	TO	TO	TO	TO	L8	TO	TO	TO	TO	TO	TO

comparable queries (all significant improvements on WSRT), which in turn better than SPLENDID in 5/7 comparable queries (all significant improvements on WSRT).

– *Large Data queries*: The most important result for large data queries is that no system can be regarded as superior because all systems are only able to produce complete results for a single query (i.e., L7). This shows the current *implementation* of query planning strategies (i.e., bushy trees in ANAPSID, left-deep trees in FedX, and dynamic programming [19] in SPLENDID) and 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 are not mature enough to deal with large data. For the only one comparable large data query, ANAPSID perform better than other systems. All of the errors thrown by the systems are available at benchmark homepage.

In a nutshell, our results clearly suggests that benchmarks with only simple queries with 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 are greatly affected once the queries goes 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.

5.3 SPARQL 1.1 Experimental Results

As per current implementations (October 2014), 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 the comparable to those presented before. Note that our SPARQL 1.1 version of the queries make 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 4 show the pure query execution performance without TPWSS.

For simple 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 is 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

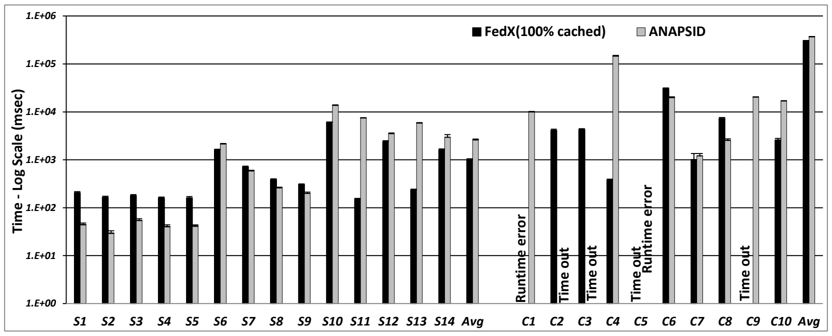


Fig. 6: Query execution time for the SPARQL 1.1 version of LargeRDFBench.

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 query category, FedX overestimates more than half (142 FedX vs. 67 optimal ref. Table 2) 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 key performance metric missing in the state-of-the-art SPARQL query federation benchmarks. Even though ANAPSID does not overestimate the relevant sources to a large extent (80 ANAPSID vs. 67 optimal), yet its performance is improved by 94% on SPARQL 1.1 versions of the simple queries. The reason is the poor query decomposition plan[8] generated for the SPARQL 1.0 version of queries S6 (308514 ms vs. 1620 ms) and S7 (298954 ms vs. 2157 ms).

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 as expected because FedX overestimated more sources than ANAPSID (199 FedX vs. 111 ANAPSID ref. Table 2). Thus, an optimal TPWSS (as in SPARQL 1.1 version of LargeRDFBench) provides more benefits to FedX. Due to the optimal source selection, FedX's performance is improved by 28.5% and ANAPSID's performance is only improved by 0.8%.

In general, the results above clearly suggests 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 allow providing tool developers with more fine-grained insights pertaining to their frameworks.

5.4 Threats to Validity

As mentioned before, The negligibly small standard deviation error bars suggest a high consistency of the presented results on local, dedicated network setup, as used in our evaluation. However, the network latency is important to be considered for query federation over live SPARQL endpoints. In particular, this factor can influence our simple queries (S1-S14) due to their small execution time (mostly in few seconds).

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

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. 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 queries that involve processing large intermediate result sets or lead to large result sets. Alarmingly, the systems return partly incomplete results without making the user aware of this incompleteness. In the future, triple stores that supports SPARQL 1.1 federated queries can also be tested with this benchmark.

References

1. G. Aluç, O. Hartig, M. T. Ozsu, and K. Daudjee. Diversified stress testing of rdf data management systems. In *ISWC*, 2014.
2. M. Arias, J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente. An empirical study of real-world SPARQL queries. *CoRR*, 2011.
3. C. Bizer and A. Schultz. The berlin sparql benchmark. *IJSWIS*, 2009.
4. S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea. Apples and oranges: A comparison of rdf benchmarks and real rdf datasets. In *SIGMOD*, 2011.
5. O. Görlitz and S. Staab. SPLENDID: Sparql endpoint federation exploiting void descriptions. In *COLD*, 2011.
6. O. Görlitz, M. Thimm, and S. Staab. Splodge: Systematic generation of sparql benchmark queries for linked open data. In *ISWC*. 2012.
7. Y. Guo and J. Heflin. LUBM: A benchmark for owl knowledge base systems. *JWS*, 2005.
8. G. Montoya, M.-E. Vidal, and M. Acosta. A heuristic-based approach for planning federated sparql queries. In *COLD*, 2012.
9. G. Montoya, M.-E. Vidal, O. Corcho, E. Ruckhaus, and C. Buil-Aranda. Benchmarking federated sparql query engines: are existing testbeds enough? In *ISWC*. 2012.
10. M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. Dbpedia sparql benchmark - performance assessment with real queries on real data. In *ISWC*, 2011.
11. F. Picalausa and S. Vansummeren. What are real sparql queries like? In *SWIM*, 2011.
12. M. Saleem, R. K. Maulik, I. Aftab, S. Shanmukha, F. D. Helena, and N. Axel-Cyrille. Fostering serendipity through big linked data. In *SWC at ISWC*, 2013.
13. M. Saleem and A.-C. Ngonga Ngomo. HiBISCuS: Hypergraph-based source selection for sparql endpoint federation. In *ESWC*, 2014.
14. M. Saleem, A.-C. Ngonga Ngomo, J. Xavier Parreira, H. Deus, and M. Hauswirth. DAW: Duplicate-aware federated query processing over the web of data. In *ISWC*, 2013.
15. M. Saleem, S. Sampath, A.-C. N. Ngomo, A. Iqbal, J. S. Almeida, S. Decker, and H. Deus. TopFed: TCGA tailored federated query processing and linking to LOD. *JBMS*, 2014.
16. M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. Fedbench: A benchmark suite for federated semantic data query processing. In *ISWC*, 2011.
17. M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. Sp2bench: a sparql performance benchmark. In *ICDE*, 2009.
18. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization techniques for federated query processing on linked data. In *ISWC*, 2011.
19. P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *ICMD*, 1979.
20. H. Wu, T. Fujiwara, Y. Yamamoto, J. Bolleman, and A. Yamaguchi. Biobenchmark toyama 2012: an evaluation of the performance of triple stores on biological data. *JBMS*, 2014.