FedShop: A Benchmark for Testing the Scalability of SPARQL Federation Engines

Minh-Hoang Dang[1](✉), Julien Aimonier-Davat[1], Pascal Molli[1], Olaf Hartig[2], Hala Skaf-Molli[1], and Yotlan Le Crom[1]

[1] Nantes Université, CNRS, LS2N, UMR 6004, 44000 Nantes, France
minh-hoang.dang@univ-nantes.fr

[2] Department of Computer and Information Science (IDA), Linköping University, Linköping, Sweden

Abstract. While several approaches to query a federation of SPARQL endpoints have been proposed in the literature, very little is known about the effectiveness of these approaches and the behavior of the resulting query engines for cases in which the number of federation members increases. The existing benchmarks that are typically used to evaluate SPARQL federation engines do not consider such a form of scalability. In this paper, we set out to close this knowledge gap by investigating the behavior of 4 state-of-the-art SPARQL federation engines using a novel benchmark designed for scalability experiments. Based on the benchmark, we show that scalability is a challenge for each of these engines, especially with respect to the effectiveness of their source selection & query decomposition approaches. FedShop is freely available online at: https://github.com/GDD-Nantes/FedShop.

Keywords: Federated Query Processing · Scalability · Source Selection · SPARQL

1 Introduction

Context and motivation: Several query engines for querying federations of SPARQL endpoints have been proposed in recent years [2,15,19]. In addition to different approaches to finding efficient query execution plans, these engines employ different source selection & query decomposition approaches. These approaches decompose any given query into subqueries associated with the federation members from which it is possible to retrieve relevant results for answering the given query. Any subquery for a federation member whose result will be either empty or cannot contribute to the overall result of the given query can be pruned in this step, reducing the effort and time needed to execute the query. Yet, the challenge is to identify such subqueries. The effectiveness of the approaches proposed for this task (and also of the query optimization approaches used by the engines) is typically evaluated using one of two benchmarks: FedBench [21] or

LargeRDFBench [18]. Both of these benchmarks are designed based on a fixed federation with a few hand-picked federation members.

Due to this design choice, these benchmarks cannot be used to study how the proposed approaches and engines behave if the number of federation members increases. Consequently, very little is known about this form of scalability of the proposed approaches and engines. While some authors have partitioned existing benchmark datasets to overcome this limitation [13,14,16,21,24], the resulting partitions do not resemble the characteristics of real-world datasets [9].

Contributions: Our main contribution in this paper is FedShop, a novel benchmark designed for scalability experiments. FedShop captures an e-commerce scenario with a scalable federation of online shops and rating sites, and with query workloads that simulate users who explore and search for products and offers across the federation. More specifically, the benchmark consists of the following:

– **10 pre-generated federations** ranging from 20 to 200 federation members,
– a schema-based **dataset generator** to generate further federations for which the scale factor is the number of federation members and for which the distribution law of every relationship of the data schema can be configured,
– **12 query templates** capturing different, use-case-specific types of queries,
– a collection of ten such queries per template (i.e., **120 queries** overall), and
– **reference source assignments** for each of these 120 queries over each of the ten pre-generated federations, as could be produced by a source selection approach that has provenance information about the complete query results.

Given the FedShop benchmark (as will be introduced in detail in Sect. 3), we make further contributions in this paper. In particular, we analyze the benchmark queries based on their Reference Source Assignments (cf. Sect. 4) and, then, present a comprehensive experimental study (cf. Sect. 5). In this study, we first show that, when using their reference source assignments, each of the 120 queries can be executed in less than 2 s over each of the 10 pre-generated federations. This illustrates that effective query decomposition and source selection can enable a customer to perform interactive queries on a federation of 200 endpoints. Thereafter, we use the benchmark to show novel experimental results that shed light on the scalability of four state-of-the-art SPARQL federation engines. The main takeaway of this study is that scalability is a challenge for the engines. None of the engines can deliver reasonable performance when querying the federation with 200 federation members. The reason for these performance issues is that the source selection approaches of the engines fail to produce source assignments that are even close to ideal. By uncovering these issues, we show that the benchmark provides an important new tool for evaluating the efficiency and scalability of approaches to query federations of RDF data sources.

The code source of the benchmark, as well as set-up instructions, further documentation and measurements of our experimental study are available online[1].

Furthermore, we provide access to the dataset, queries, and Virtuoso dump generated for experimentation.[2]

[1] https://github.com/GDD-Nantes/FedShop.
[2] https://doi.org/10.5281/zenodo.7919872.

In addition to using it for the experimental study presented in this paper, we contributed a more limited fragment of FedShop (only 12 queries and two federations) as a use case for a recent hackathon[3] on query federation. During the hackathon, developers of query federation engines used this fragment of FedShop to reveal several implementation issues and weaknesses in their engines.

2 Related Works

The benchmark FedBench [21] is mainly used for testing and analyzing the performance of SPARQL federation engines processing RDF hosted in SPARQL endpoints. The dataset of FedBench is a collection of 10 datasets from different domains: 4 datasets from life-sciences (Kegg, Chebi,...), 6 cross domains datasets (Dbpedia,Geonames,...), and 14 real queries. To add more datasets and queries, FedBench proposes a setup FedBench/SP2Bench with a collection of 16 datasets generated by clustering of the SP2Bench dataset [22] and 11 queries from SP2Bench. To transform SP2Bench into a federated benchmark, FedBench applied a clustering on the types of the SP2Bench dataset, i.e., each class of SP2Bench (Person, Article, Inproceedings,...) is assigned to different SPARQL endpoints, which generates 16 synthetic datasets. The main issue with this approach is that clustering on classes is limited to 16, and "the partitions do not resemble the characteristics of real-world datasets", as pointed out in [9].

LargeRdfBench [11,18] extends the dataset and workload of FedBench, respectively, to 13 datasets and 40 queries. However, FedBench, FedBench/SP2Bench, and LargeRdfBench do not allow the evaluation of the behavior of the federated query engines when the number of federation members increases. It is also possible to add more federated queries as proposed in QFed [17]. However, having more federated queries does not address the issue of scalability with the number of federation members.

To scale on the federation size, Lusail [1], DARQ [16], LHD [24], LILAC [13] propose partitioning datasets into several datasets. The different techniques for partitioning a dataset is reported in [14]; it includes horizontal, vertical, and hybrid partitioning with or without replication. First, as highlighted in [8], synthetic benchmarks such as BSBM [6], LUBM [10], or SP2Bench [22] can be highly structured and do not correspond to more realistic data hosted on a public endpoint. Partitioning/clustering such synthetic data does not solve the problem of structuredness [8].

In LHD [24], BSBM [6] entities (products, offers, producers,...) have been hashed on their subjects and distributed over 10 SPARQL endpoints. Such partitioning is working but does not correspond to a real use-case. To evaluate DARQ [16], the LUBM benchmark [10] dataset has been partitioned with LUBM classes, generating the same issues as FedBench/SP2Bench. In Lusail [1], an evaluation has been conducted with 256 universities generated using the LUBM benchmark and distributed over 256 SPARQL endpoints. Interlinks come from professors who worked in different universities and students who graduated from

[3] https://github.com/MaastrichtU-IDS/federatedQueryKG.

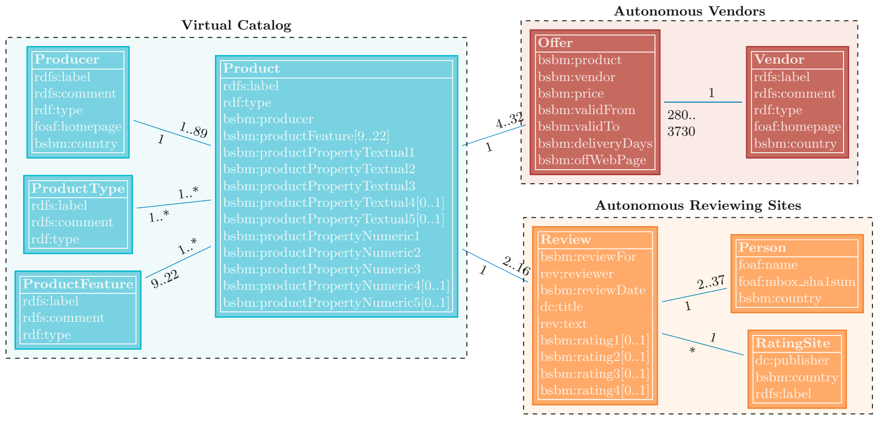


Fig. 1. The overall schema of BSBM extracted from [6]

different universities. Given these relations, 3 queries from the LUBM benchmark can be executed as federated queries (Q2, Q9, and Q13). In this setup, it is possible to increase the number of sources by adding new universities. Although this setup is interesting, the LUBM benchmark is designed for evaluating the reasoning capabilities of SPARQL engines, not with a real-use case in mind as with the explore use-case of Berlin Benchmark, for example. Consequently, a few queries make sense in a federated context and are not very challenging for a federated evaluation. In FedShop, we follow the same approach as Lusail, but we start from Berlin Benchmark and follow the explore use-case of BSBM.

3 The FedShop Benchmark

The FedShop use-case, inspired by the Berlin Benchmark (BSBM) use-case [6], involves a customer navigating through a virtual shop that comprises multiple autonomous shops, each with its own SPARQL endpoint. This exploration is powered by SPARQL queries executed across the federation, giving the illusion of one endpoint hosting all the different vendors as in the original BSBM. The SPARQL queries primarily aim to retrieve products based on certain criteria, obtain more product information, compare products, find similar products, and locate product reviews. Customers need to receive real-time feedback when using the FedShop use-case, which means all queries must processed quickly, within a few seconds.

The scalability of the benchmark is obtained by incorporating more shops (or reviewing sites) into the federated shop.

3.1 FedShop Data Generation

The overall schema of RDF data follows the schema described in Fig. 1. This schema is as close as possible to the schema of BSBM. We consider 3 different components in the schema. The virtual catalog comprises the products and their features, and it is shared by vendors and reviewing sites.

Each member of the federation (whether a vendor or a rating site) is considered autonomous and capable of operating independently of other members. Specifically, all product or review queries must generate results based solely on the vendor or rating site being queried. To ensure the autonomy of each member, we adhere to a straightforward guideline: *If a shop sells a product, it will have a local URI for that product, and all related information can be accessed via local URLs*. Essentially, this replicates how producers represent their products in the vendor domain. We follow the same approach for rating sites.

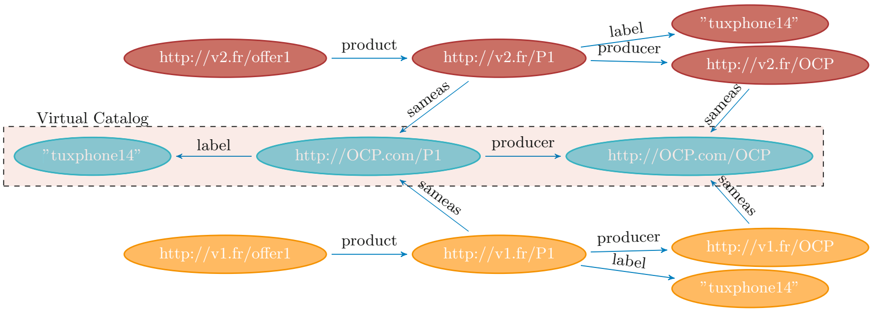


Fig. 2. Virtual catalog and replication of products across vendors

As entities of type Product, ProductType, Producer, and ProductFeature are replicated on many sites, all federation members have a `sameAs` link from local entities to the global entities of the virtual catalog. To illustrate, suppose two vendors $V1$ and $V2$ are selling the same product $P1$ named "tuxphone14" produced by the "OCP" company. Figure 2 describes the triples hosted by $V1$ on http://v1.fr and $V2$ on http://v1.fr. As we can see, products and their descriptions are replicated with local URLs by each vendor. `sameAs` links keep the connection with products of the virtual catalog. Following the FedShop data generation rules, we are sure that all the subjects of a vendor or a reviewing site are specific to its web domain, i.e., two different vendors cannot share the same subjects. We also know that all objects of `sameAs` predicates are global, i.e., potentially shared by all endpoints.

Generating a federated shop comprises two steps:

1. First, generate a virtual catalog of products shared by all vendors and rating sites. This catalog has a fixed size and will not be part of the final data.
2. Second, generate vendors and rating sites, each replicating products from the virtual catalog. The replication process follows a distribution law that decides how often the same product may appear on different vendors/rating sites.

To control the distribution laws when creating catalogs, vendors and rating sites, we rely on schema-based data generators such as WatDiv [4] or gMark [5]. As schema and distribution laws are declared as part of a specification, schema-generators allows to change the schema and distribution laws easily.

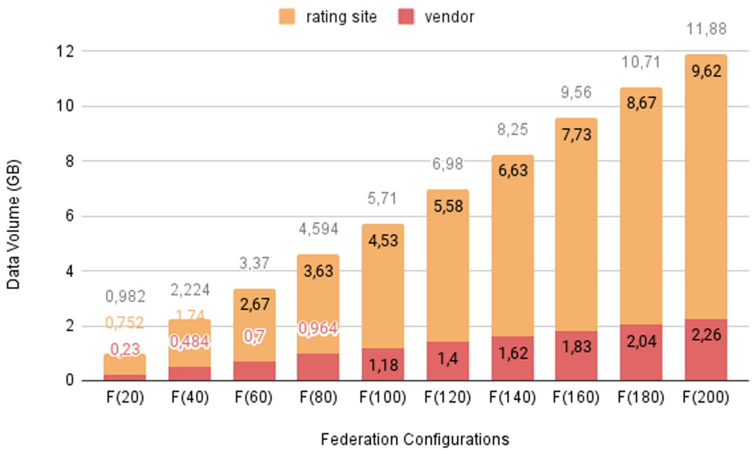
For the catalog, we chose to approximate the original BSBM configuration with 200,000 products. With this catalog, we generated a first federation of 20 sources $F(20)$ composed of 10 vendors and 10 rating sites. Next, we generated $F(40)$ just by adding 10 new vendors and 10 new rating sites; therefore, $F(40)$ includes $F(20)$. We continue this process until we reach $F(200)$. We described the overall size of the Federation in Table 1. It should be emphasized that the catalog itself does not constitute a component of the federation. Since all products are duplicated across all federation members, maintaining the catalog is unnecessary.

3.2 FedShop Query Generation

We follow the explore use-case of BSBM composed of 12 template queries. The explore use-case simulates a scenario where a customer is searching for products and reviews, e.g., *find products for a given set of generic features, retrieve basic information about a specific product, find products that are similar to a given product, retrieve in-depth information about a specific product, including offers and reviews.*

Table 1. Data Volume (#quads and storage) across FedShop sources. The bar chart breaks down the size of each federation by category.

	F(10)	**F(20)**	**F(30)**	**F(40)**	**F(50)**	**F(60)**	**F(70)**	**F(80)**	**F(90)**	**F(100)**
nquads (M)	5.167	11.821	17.852	24.335	30.374	37.080	43.827	50.764	56.883	63.079
size GB	0.98	2.21	3.35	4.57	5.71	6.97	8.25	9.56	10.71	11.88



Template Federated Queries. We transformed the BSBM template queries to reflect the federated nature of the data in FedShop. For instance, the template query $Q1$ of BSBM allows a consumer to search for a product by knowing the product type and 2 product features. The template query $Q1$ has 4 placeholders: %ProductType%, %ProductFeature1%, %ProductFeature2%, %x%, which must be instantiated to be executed.

Listing 1 describes how we transformed the original Q1 template query of BSBM (Listing 3a) into the Q1 template query of FedShop (Listing 3b). Since local entities are linked to global entities, as shown in Sect. 3.1, we need to add a new `sameAs` triple pattern in the Q1 FedShop template query to link local products to global products. The principle of autonomy enables the execution of FedShop queries on SPARQL endpoints of any member, thereby returning results specific to that member. The same queries can also be executed in federation mode, thus transforming all vendors/rating sites into a virtual federated shop.

We applied this strategy to transform the 12 template queries of BSBM into 12 FedShop template queries.

Instantiate Template Federated Queries. To produce executable queries, we need to replace placeholders in template queries with real values. The objective is to find combinations of values with various selectivities that return results for all configurations of FedShop.

```
SELECT DISTINCT ?product ?label
WHERE {
 ?product rdfs:label ?label .
 ?product a %ProductType% .
 ?product bsbm:productFeature %ProductFeature1% .
 ?product bsbm:productFeature %ProductFeature2% .
 ?product bsbm:productPropertyNumeric1 ?value1 .
 FILTER (?value1 > %x%)
}
ORDER BY ?label
LIMIT 10
```

(a) Q1 BSBM query

```
SELECT DISTINCT ?product ?label
WHERE {
    ?product rdfs:label ?label .

    ?product rdf:type ?localProductType .
    ?localProductType owl:sameAs %ProductType% .

    ?product bsbm:productFeature ?localProductFeature1 .
    ?localProductFeature1 owl:sameAs %ProductFeature1% .

    ?product bsbm:productFeature ?localProductFeature2 .
    ?localProductFeature2 owl:sameAs %ProductFeature2% .
    ?product bsbm:productPropertyNumeric1 ?value1 .
    FILTER (?value1 > %x%)
}
ORDER BY ?label
LIMIT 10
```

(b) Q1 FedShop query

Lst. 1. Transformation of the Q1 BSBM template query into the Q1 FedShop template query

The overall process to instantiate template queries is as follows: (1) rewrite template queries as generic queries where placeholders are replaced with variables, (2) execute this queries on the $F(20)$ configuration, (3) chooses a distinct combination of values for placeholders randomly among the results of the execution.

For each of the 12 template queries, we generated 10 instances on the $F(20)$ configuration, each with a randomly selected combination of values We adopted this process for 2 reasons:

1. By instantiating on $F(20)$ we guarantee that the number of results of query instances can only grow when the number of sources increases, i.e., for configuration $> F(20)$.
2. By randomly selecting 10 combinations of values, we guarantee diversity w.r.t queries selectivity when the number of sources increases. For instance, the selectivity of a combination of placeholders can be different for $F(20)$ and $F(200)$. Since we do not know when generating $F(20)$ how the selectivity will evolve, we randomly choose 10 combinations of placeholders and observe experimentally how it evolves.

This entire process produces a workload of 120 queries that can run on 10 federations: $F(20)$ to $F(200)$. As in FedBench [21], we analyzed the structural features of our workload in Table 2. In sum, the FedShop queries comprise 1–18 triple patterns, with `UNION`, `OPTIONAL`, and many filters, including Regexp. Many queries use `ORDER BY`/`LIMIT` clauses along with `DISTINCT` projections.

4 FedShop Reference Source Assignment (RSA)

For each instantiated query, we generated a Reference Source Assignment (RSA) standard, i.e., an executable SPARQL 1.1 query with service clauses where source assignment has been precomputed [7]. Producing RSA queries follow two important objectives:

Table 2. Fedshop queries features: #tp: Number of triple patterns, #bgp: Number of BGP, #join: Number of joins, #union: Number of unions, #var: Number of variables in a query #nodes: Number of nodes in the graph representation of the query, #edge: Number of edges in the graph representation of the query, MJVD: mean join vertices degree, #OPT: Number of OPTIONAL, #regex: FILTER regexp, #filterR: FILTER relational, OB: order by.

query	#tp	#bgp	#join	#union	#var	#nodes	#edges	MJVD	#OPT	#regex	#filterR	OB	Distinct	Limit
q01	8	1	7	0	6	9	8	1,78	0	0	1	1	1	1
q02	16	4	12	0	16	17	16	1,88	3	0	0	0	0	0
q03	11	2	9	0	9	12	11	1,83	1	0	2	1	0	1
q04	18	2	16	1	9	13	12	1,85	0	0	2	1	1	1
q05	11	1	10	0	11	12	11	1,83	0	0	5	1	1	1
q06	2	1	1	0	2	3	2	1,33	0	1	0	0	0	0
q07	17	5	12	0	15	18	17	1,89	4	0	1	0	0	0
q08	11	5	6	0	11	12	11	1,83	4	0	0	1	0	1
q09	1	1	0	0	1	2	1	1,00	0	0	0	0	0	0
q10	7	1	6	0	6	8	7	1,75	0	0	2	1	1	1
q11	3	3	1	1	4	4	3	1,50	0	0	0	0	0	0
q12	10	1	9	0	10	11	10	1,82	0	0	0	0	0	0

1) Verifying that all FedShop queries can be executed with reasonable execution time, so the use-case of exploring a federated shop is realizable, and 2) observing the gap of performances of existing federated query engines.

To produce the Reference Source Assignment, we proceed in two steps:

(i) We perform query decomposition queries manually into SPARQL 1.1 subqueries with `SERVICE` clauses [2] and exhaustive source assignment, i.e., each subquery is initially assigned to all federation members. We perform query decomposition by analyzing the join variables of queries, i.e., variables used by at least 2 triple patterns. Because of the FedShop data generation rules described in Sect. 3.1, we know that join variables on subjects can only be resolved on one endpoint. This corresponds to the notion of local join variables in [1]. We also know that join variables that need to be resolved between two endpoints can only appear as objects of `sameAs` predicates or as an attribute. This corresponds to the concept of global join variables in [1]. As there is no query with join variables on attributes, global join variables can only appear as objects of `sameAs` predicates. Through the analysis of the join variables, we can find an efficient decomposition of template queries.

(ii) We compute the minimal source selection over decomposed queries with a simple query rewriting and an evaluation over the union of all data.

By analyzing the join variables of the 12 template queries, we observed three kinds of decomposition:

Single-domain Decomposition is applied to queries with no global join variables and where a subject of a triple pattern is bound as the query Q12 presented in Listing 3a (the bounded subject is v7:Offer858). For these queries, all triple patterns can be grouped into one service clause, and only one endpoint should return results. Consequently, such a query can be rewritten with only one service clause, as presented in Listing 3b. With no further informa-

```
prefix v7: <http://www.vendor7.fr>
SELECT * WHERE { #Q12
 v7:Offer858 bsbm:product ?productURI .
 ?productURI owl:sameAs ?ProductXYZ .
 ?productURI rdfs:label ?productlabel .
 v7:Offer858 bsbm:vendor ?vendorURI .
 ?vendorURI rdfs:label ?vendorname .
 ?vendorURI foaf:homepage ?vendorhomepage .
 v7:Offer858 bsbm:offerWebpage ?offerURL .
 v7:Offer858 bsbm:price ?price .
 v7:Offer858 bsbm:deliveryDays ?deliveryDays .
 v7:Offer858 bsbm:validTo ?validTo }
```

(a) Single Domain Query (Q12)

```
prefix v7: <http://www.vendor7.fr>
SELECT DISTINCT * WHERE { #Q12
 VALUES ?bgp1 { <http://vendor1.fr> <...> }
 SERVICE ?bgp1 {
 <v7/Offer858> bsbm:product ?productURI .
 ?productURI owl:sameAs ?ProductXYZ .
 ?productURI rdfs:label ?productlabel .
 <v7/Offer858> bsbm:vendor ?vendorURI .
 ?vendorURI rdfs:label ?vendorname .
 ?vendorURI foaf:homepage ?vendorhomepage .
 <v7/Offer858> bsbm:offerWebpage ?offerURL .
 <v7/Offer858> bsbm:price ?price .
 <v7/Offer858> bsbm:deliveryDays ?deliveryDays .
 <v7/Offer858> bsbm:validTo ?validTo }}
```

(b) Single Domain service query (Q12)

Lst. 2. Single Domain Queries

```
SELECT DISTINCT ?product ?label
WHERE {
 ?product rdfs:label ?label .
 ?product rdf:type ?localProductType .
 ?localProductType owl:sameAs bsbm:ProductType647 .
 ?product bsbm:productFeature ?localProductFeature1 .
 ?localProductFeature1 owl:sameAs bsbm:ProductFeature8774 .
 ?product bsbm:productFeature ?localProductFeature2 .
 ?localProductFeature2 owl:sameAs bsbm:ProductFeature16935 .
 ?product bsbm:productPropertyNumeric1 ?value1 .
  FILTER (?value1 > "744"^^xsd:integer) }
ORDER BY ?label
LIMIT 10
```

(a) Multi Domain Query (Q1)

```
SELECT DISTINCT ?product ?label WHERE {
 VALUES ( ?bgp1 ) { <http://www.vendor1.fr/> <...> }
 SERVICE ?bgp1 {
 ?product rdfs:label ?label .
 ?product rdf:type ?localProductType .
 ?localProductType owl:sameAs bsbm:ProductType647 .
 ?product bsbm:productFeature ?localProductFeature1 .
 ?localProductFeature1 owl:sameAs bsbm:ProductFeature8774 .
 ?product bsbm:productFeature ?localProductFeature2 .
 ?localProductFeature2 owl:sameAs bsbm:ProductFeature16935 .
 ?product bsbm:productPropertyNumeric1 ?value1 .
   FILTER (?value1 > "744"^^xsd:integer) }}
ORDER BY ?product ?label
LIMIT 10
```

(b) Multi Domain service query (Q1)

Lst. 3. Multi-domain Queries: Q1

tion, all endpoints have to be considered to find the one that contains results. Queries Q9, Q11, and Q12 belong to the single domain class of decomposition.

Multi-domain Decomposition is applied to queries with no global join variables and no bounded subject in triple patterns as the query Q1 in Listing 3a. For these queries, all the triple patterns can be grouped into one service clause, but multiple endpoints may return results. Listing 3b describes the decomposition of query Q1. With no additional information, all endpoints have to be considered to find those that return results. Queries Q1, Q2, Q3, Q4, Q6, Q8, and Q10 belong to the multi-domain class of decomposition.

Cross-domain Decomposition is applied to queries with global join variables as the query Q5 described in Listing 3a. Q5 has 2 global join variables: `?product` and `?productFeature`. Following the decomposition algorithm of

```
SELECT DISTINCT ?product ?localProductLabel
WHERE {
 ?localProduct rdfs:label ?localProductLabel .
 ?localProduct bsbm:productFeature ?localProdFeature .
 ?localProduct bsbm:productPropertyNumeric1 ?simProperty1 .
 ?localProduct bsbm:productPropertyNumeric2 ?simProperty2 .
 ?localProduct owl:sameAs ?product .
 ?localProdFeature owl:sameAs ?prodFeature .
 ?localProductXYZ bsbm:productFeature ?localProdFeatureXYZ .
 ?localProductXYZ bsbm:productPropertyNumeric1 ?origProperty1 .
 ?localProductXYZ bsbm:productPropertyNumeric2 ?origProperty2 .
 ?localProductXYZ owl:sameAs bsbm:Product136030 .
 ?localProdFeatureXYZ owl:sameAs ?prodFeature .
  FILTER (bsbm:Product136030 != ?product)
  FILTER (?simProperty1 < (?origProperty1 + 20) &&
   ?simProperty1 > (?origProperty1 - 20))
  FILTER (?simProperty2 < (?origProperty2 + 70) &&
   ?simProperty2 > (?origProperty2 - 70))}
ORDER BY ?localProductLabel
LIMIT 5
```

(a) Cross Domain Query (Q5)

```
SELECT DISTINCT ?product ?localProductLabel WHERE {
 VALUES ( ?bgp1 ?bgp2 ) {
 ( <http://www.vendor1.fr/> <http://www.vendor1.fr/> )
 ( <http://www.vendor1.fr/> <http://www.vendor2.fr/> )
 # ... ) }
SERVICE ?bgp1 {
 ?localProductXYZ owl:sameAs bsbm:Product136030 .
 ?localProductXYZ bsbm:productFeature ?localProdFeatureXYZ .
 ?localProdFeatureXYZ owl:sameAs ?prodFeature .
 ?localProductXYZ bsbm:productPropertyNumeric1 ?origProperty1 .
 ?localProductXYZ bsbm:productPropertyNumeric2 ?origProperty2} .
SERVICE ?bgp2 {
 ?localProduct owl:sameAs ?product .
  FILTER (bsbm:Product136030 != ?product)
 ?localProduct rdfs:label ?localProductLabel .
 ?localProduct bsbm:productFeature ?localProdFeature .
 ?localProdFeature owl:sameAs ?prodFeature .
 ?localProduct bsbm:productPropertyNumeric1 ?simProperty1 .
 ?localProduct bsbm:productPropertyNumeric2 ?simProperty2} .
FILTER(?simProperty1 < (?origProperty1 + 20) &&
  ?simProperty1 > (?origProperty1 - 20))
FILTER(?simProperty2 < (?origProperty2 + 70) &&
  ?simProperty2 > (?origProperty2 - 70))}
ORDER BY ?product ?localProductLabel
LIMIT 5
```

(b) Cross-Domain service query (Q5)

Lst. 4. Cross-Domain Queries: Q5

[1], we generate two *exclusive groups* one related to `?localProductXYZ` and the second is related to `?localProduct`. For this decomposition, we generate 2 service clauses as described in Listing 3b. A filter push-down allows to push one filter into a service clause. In this decomposition, we consider that all combinations of pairs of endpoints should be contacted, i.e., all pairs of sources should be declared in the `VALUES` clause of the service query. Queries Q5 and Q7 belong to the cross-domain class of decomposition.

Until now, the query decomposition and source selection produce an exhaustive source assignment, i.e., all combinations of endpoints are considered. For computing minimal source assignment [7], we rewrite service queries into *provenance queries* by simply replacing `SERVICE` clauses with `GRAPH` clauses and changing the projection of the query as illustrated in Fig. 5 for the query Q5. The execution of the query Q5prov over the union of RDF data of all federation members returns the pairs of endpoints that effectively contribute to the results of query Q5. The results of provenance queries are used to update the `VALUES` clauses of RSA queries.

To validate our query decomposition and source selection, we verified that all RSA queries return correct and complete results, i.e., we obtain the same results as original queries evaluated over the union of all datasets. We also verified that RSA queries could be executed under 2 s for all configurations of FedShop (see Sect. 5).

```
SELECT DISTINCT ?product ?localProductLabel WHERE {
 VALUES ( ?bgp1 ?bgp2 ) {
 ( <http://www.vendor1.fr/> <http://www.vendor1.fr/> )
 ( <http://www.vendor1.fr/> <http://www.vendor2.fr/> )
 # ... ) }
SERVICE ?bgp1 {
 ?localProductXYZ owl:sameAs bsbm:Product136030 .
 ?localProductXYZ bsbm:productFeature ?localProdFeatureXYZ .
 ?localProdFeatureXYZ owl:sameAs ?prodFeature .
 ?localProductXYZ bsbm:productPropertyNumeric1 ?origProperty1 .
 ?localProductXYZ bsbm:productPropertyNumeric2 ?origProperty2} .
SERVICE ?bgp2 {
 ?localProduct owl:sameAs ?product .
  FILTER (bsbm:Product136030 != ?product)
 ?localProduct rdfs:label ?localProductLabel .
 ?localProduct bsbm:productFeature ?localProdFeature .
 ?localProdFeature owl:sameAs ?prodFeature .
 ?localProduct bsbm:productPropertyNumeric1 ?simProperty1 .
 ?localProduct bsbm:productPropertyNumeric2 ?simProperty2} .
FILTER(?simProperty1 < (?origProperty1 + 20) &&
  ?simProperty1 > (?origProperty1 - 20))
FILTER(?simProperty2 < (?origProperty2 + 70) &&
  ?simProperty2 > (?origProperty2 - 70))}
ORDER BY ?product ?localProductLabel
LIMIT 5
```

(a) Cross-Domain service query $Q5$

```
SELECT DISTINCT ?bgp1 ?bgp2 WHERE {
graph ?bgp1 {
 ?localProductXYZ owl:sameAs bsbm:Product136030 .
 ?localProductXYZ bsbm:productFeature ?localProdFeatureXYZ .
 ?localProdFeatureXYZ owl:sameAs ?prodFeature .
 ?localProductXYZ bsbm:productPropertyNumeric1 ?origProperty1 .
 ?localProductXYZ bsbm:productPropertyNumeric2 ?origProperty2} .
graph ?bgp2 {
 ?localProduct owl:sameAs ?product .
  FILTER (bsbm:Product136030 != ?product)
 ?localProduct rdfs:label ?localProductLabel .
 ?localProduct bsbm:productFeature ?localProdFeature .
 ?localProdFeature owl:sameAs ?prodFeature .
 ?localProduct bsbm:productPropertyNumeric1 ?simProperty1 .
 ?localProduct bsbm:productPropertyNumeric2 ?simProperty2} .
FILTER(?simProperty1 < (?origProperty1 + 20) &&
  ?simProperty1 > (?origProperty1 - 20))
FILTER(?simProperty2 < (?origProperty2 + 70) &&
  ?simProperty2 > (?origProperty2 - 70))}
ORDER BY ?product ?localProductLabel
LIMIT 5
```

(b) Provenance query for $Q5_{prov}$

Lst. 5. Source selection for an instance of $Q5$

5 Experimental Study

The experimental study aims to answer the following questions:

(1) What is the performance of the Reference Source Assignment (RSA) defined in Sect. 4? Is it possible to run a federated shop of 200 shops with federated queries? (2) How do the performances of existing federated query engines as the federation size increases? What is the gap with the RSA ?

Generating Data and Queries. We used the FedShop data generator to generate a catalog of 200 000 products following the schema of Fig. 1. Next, we generated 10 federations: F(20) to F(200), where each federation is composed of half of the vendors and half of the reviewing sites, i.e., F(200) is a federation of 100 vendors and 100 review sites. All instructions to install, configure, and run the FedShop benchmark are available on the FedShop GitHub repository. We used the FedShop query generator to instantiate a workload of 120 queries, where each template query is randomly instantiated 10 times. The 120 queries are to be executed on each of the 10 federations: F(20) to F(200). Detailed statistics on the generated data are available in the Jupyter Notebook of the repository.

Setting Up Federations. Setting up a federation of 200 endpoints raises serious questions about the tractability of the experiment. Starting one physical endpoint per vendor/reviewing as proposed in KOBE [12] is not realistic if one considers large federations. We take another approach based on Virtual Endpoints as proposed in Virtuoso. All endpoints are represented as Virtual Endpoints hosted in one Virtuoso server. Each endpoint is connected to a named RDF graph corresponding to one vendor or one reviewing site[4]. The monitored federated query engine is not aware that all endpoints are virtual. The number of threads of the Virtuoso server is 20, and those of the federated query engines are 20; they are defined so that all subqueries of a federated query engine are executed in parallel on the Virtuoso server. Each query in the workload has to be executed sequentially to get the correct measurement of execution times.

To run the FedShop benchmark, we developed the FedShop runner that automatically deploys the federations, runs the queries among the different configurations, and monitors the federated query engines under evaluation. All the instructions to run the benchmark and add a new federated query engine are available on the FedShop GitHub repository.

Evaluated Engines. To validate the benchmark's ability to analyze different engines and to reveal a new class of insights about these engines, we run the 10 configurations of FedShop on engines assessed in CostFed [20], namely: FedX [23], CostFed [20], ANAPSID [3] and SPLENDID [9]. Lusail [1] is not part of the evaluation because no implementation is available. We run the RSA queries with Apache Jena. All evaluated engines were integrated into the FedShop runner to ensure the reproducibility of results. We relaunch the federated query engine each time we run a query, i.e., federated query caches are not kept between the processing of 2 different queries.

[4] The Virtuoso database used at generation time is reused for execution.

Metrics. In our experiment, we mainly measure the evolution of 3 metrics when the number of sources increases: (1) The *Execution Time* is the total time spent by the federated query engine to produce the query results. Each query is executed 4 times, and the reported execution time is the average of the 4 execution time. (2) The number of queries that timeout. (3) The number of queries that terminate with errors.

We set the time out to 60 s per federated query. As the FedShop use-case corresponds to an interactive exploration of shops and reviews, we consider that a user does not wait for more than 60 s for the results of any queries of FedShop. In the worst case, testing a federated engine requires 4800*60 s = 80h, 3 d.

Since the number of sources increases, if a query timeout for a federation $F(i\text{-}1)$, it should also timeout for federation $F(i)$. To save experiment time, we execute a query in $F(i)$ only if it did not timeout in $F(i\text{-}1)$ and if there is no timeout in other attempts in $F(i)$.

A query engine may also produce errors when executing a query. These errors can be caused by unsupported query features or simple runtime exceptions as out-of-memory. In case of an error, we report the error, but when computing average execution times, we attribute the timeout value to the query execution time. This means that having errors when computing the average execution time degrades the average execution time.

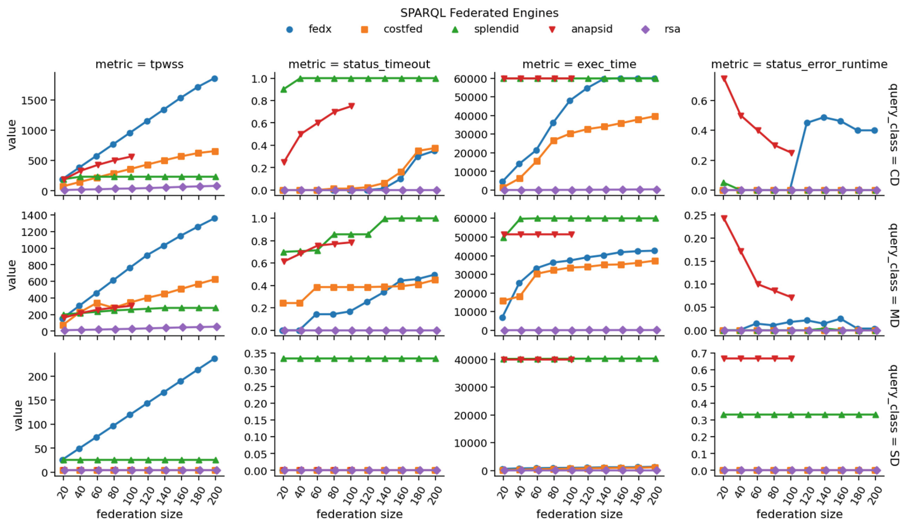


Fig. 3. Evolution of engine performance per query class. Each line of the plot corresponds to a class of queries from top to bottom: Cross-Domain (CD), Multi-Domain (MD), and Single Domain (SD). Columns correspond to evaluation metrics

5.1 Experimental Results

The overall results of the experimentation are displayed in Fig. 3. More detailed results per query and per engine are available in the Jupyter Notebook of the FedShop repository.

To improve the readability of the evaluation, we split the results according to the different classes of query decomposition detailed in Sect. 4, i.e., Single-Domain (SD) with 3 template queries and 30 instantiated queries, Multi-Domain (MD) with 7 template queries and 70 instantiated queries and Cross-Domain (CD) with 2 template queries and 20 instantiated queries.

For each class (CD/MD/SD), we computed the evolution of 3 metrics on the Y-axis as the size of the federation increases on the X-axis:

exec_time is the average execution time per query per class. The maximum value is 60 s as the timeout is set to 60 s.

error is the number of queries that finish with errors in the class. As there are 4 measurements per query, the maximum value for errors is respectively: 20 x 4 = 80 for the CD class, 70 x 4 = 280 for the MD class, and 30 x 4 = 120 for the SD class.

timeout is the number of queries that time out in the class. The maximum value per class is the same as for the error metric.

Since the size of the federation increases, source selection, execution times, and timeout should monotonically increase. The number of errors is not predictable.

We focus first on the execution time plot. We first see that RSA can execute all queries in less than 1 s on average with no errors. If we look at the detailed results in the Jupyter Notebook, the longest query of the RSA takes less than 2 s. This demonstrates that an adequate query decomposition and source selection can support interactive querying of a federation of 200 shops by a customer.

Regarding the execution time of evaluated federated engines, *we first observe that none can support interactive exploration of a federation of 200 shops.* Indeed, the best average execution time for Multi-domain queries is obtained by CostFed with more than 30 s. It goes up to 40 s for cross-domain queries. Only Single Domain queries are correctly processed by CostFed and FedX with performances similar to RSA queries. The execution times of Anapsid and Splendid increase quickly to reach the maximum average time of 60 s, i.e., all queries timeout.

We observe similar behavior on the timeout curves. No engine can terminate the workload without a timeout. FedX and CostFed process the SD queries correctly, but the number of timeouts grows quickly for MD and CD queries. Other engines have a high number of timeouts for the first configuration of FedShop.

Regarding the errors, as federated query engines are exposed for the first time to many endpoints with quite complex queries, they reveal some bugs in implementations. However, most errors come from out-of-memory errors. When federated queries are not properly decomposed, they generate significant data shipping from endpoints to federated query engines, and memory is quickly exhausted. We observe that the number of errors is decreasing for some engines.

The explanation is that the query timeouts before producing an error when the federation size increases.

Overall, we observe a significant performance gap between RSA and federated engines. Current federated engines fail to find the query decomposition of RSA. If triple patterns are not properly grouped, the number of subqueries sent to endpoints is high as the number of intermediate results transferred from endpoints to federated query engines. This seriously degrades performances. With RSA, the number of subqueries sent to endpoints is low as the number of intermediate results. This is the key to the high performance of RSA.

6 Concluding Remarks

FedShop is the first benchmark designed for studying the scalability of query federation engines. It is based on a well-known use-case of the semantic web community that makes sense in a federated context. The FedShop generator is highly configurable and generates a Reference Source Assignment standard (RSA) that can be used as an independent baseline. The FedShop runner allows running the experiment with reasonable resources in a reasonable time. It can be easily extended to integrate new federated query engines, as well as new use-cases (data and queries). We provided a smaller portion of FedShop (comprising only 12 queries and two federations) as a practical example of a recent query federation hackathon[5]. During the hackathon, the query federation engine developers utilized this FedShop fragment to identify various implementation problems and shortcomings in their engines[6]. In this paper, we presented a larger experiment highlighting the scalability issues for federated query engines and introduced the Reference Source Assignments (RSA). The RSA reveals ample opportunities for improvement in federated query engines.

As future work, considering its flexibility, FedShop can be improved in many different ways. Data generation can be customized to introduce diversity in shops where shops can have more products than others or shops are specialized in one category of product. We can also customize distribution laws per shop to control the structuredness of generated data [8]. It is also possible to introduce some noise during catalog replication to check semantic heterogeneity. For this paper, we keep BSBM queries and adapt them to the federated context. Adding new queries to have a more diversified query workload is possible. The last perspective of FedShop is beyond the benchmark. Current federation engines mainly targeted data integration use-case. A larger perspective for FedShop is to investigate how federation engines can be used to power federated applications.

Acknowledgments. This work is supported by the French ANR project DeKaloG (Decentralized Knowledge Graphs), ANR-19-CE23-0014, CE23 - Intelligence artificielle, the French CominLabs project MikroLog (The Microdata Knowledge Graph), and by Vetenskapsrådet (the Swedish Research Council, project reg. no. 2019-05655)

[5] https://github.com/MaastrichtU-IDS/federatedQueryKG.

[6] https://github.com/MaastrichtU-IDS/federatedQueryKG/blob/main/UneditedReport.pdf.

References

1. Abdelaziz, I., Mansour, E., Ouzzani, M., Aboulnaga, A., Kalnis, P.: Lusail: a system for querying linked data at scale. Proc. VLDB Endow. **11**(4), 485–498 (2017)
2. Acosta, M., Hartig, O., Sequeda, J.: Federated RDF query processing. In: Sakr, S., Zomaya, A.Y. (eds.) Encyclopedia of Big Data Technologies, pp. 754–761. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-77525-8_228
3. Acosta, M., Vidal, M.-E., Lampo, T., Castillo, J., Ruckhaus, E.: ANAPSID: an adaptive query processing engine for SPARQL endpoints. In: Aroyo, L., et al. (eds.) ISWC 2011. LNCS, vol. 7031, pp. 18–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25073-6_2
4. Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified stress testing of RDF data management systems. In: Mika, P., et al. (eds.) ISWC 2014. LNCS, vol. 8796, pp. 197–212. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11964-9_13
5. Bagan, G., Bonifati, A., Ciucanu, R., Fletcher, G.H.L., Lemay, A., Advokaat, N.: gmark: schema-driven generation of graphs and queries. In: 33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, 19–22 April 2017, pp. 63–64. IEEE Computer Society (2017)
6. Bizer, C., Schultz, A.: The berlin SPARQL benchmark. Int. J. Semantic Web Inf. Syst. **5**(2), 1–24 (2009)
7. Cheng, S., Hartig, O.: Fedqpl: a language for logical query plans over heterogeneous federations of RDF data sources. In: Indrawan-Santiago, M., Pardede, E., Salvadori, I.L., Steinbauer, M., Khalil, I., Kotsis, G. (eds.) Proceedings of the 22nd International Conference on Information Integration and Web-Based Applications & Services, Virtual Event (iiWAS 2.20)/Chiang Mai, 30 November–2 December 2020, pp. 436–445. ACM (2020)
8. Duan, S., Kementsietsidis, A., Srinivas, K., Udrea, O.: Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. In: Sellis, T.K., Miller, R.J., Kementsietsidis, A., Velegrakis, Y. (eds.) Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2011), Athens, Greece, 12–16 June 2011, pp. 145–156. ACM (2011)
9. Görlitz, O., Staab, S.: SPLENDID: SPARQL endpoint federation exploiting VOID descriptions. In: Hartig, O., Harth, A., Sequeda, J.F. (eds.) Proceedings of the Second International Workshop on Consuming Linked Data (COLD2011), Bonn, 23 October 2011, vol. 782 of CEUR Workshop Proceedings. CEUR-WS.org (2011)
10. Guo, Y., Pan, Z., Heflin, J.: LUBM: a benchmark for OWL knowledge base systems. J. Web Semant. **3**(2–3), 158–182 (2005)
11. Hasnain, A., Saleem, M., Ngonga Ngomo, A.-C., Rebholz-Schuhmann, D.: Extending largerdfbench for multi-source data at scale for SPARQL endpoint federation. In: Demidova, E., Zaveri, A., Simperl, E. (eds.) Emerging Topics in Semantic Technologies - ISWC 2018 Satellite Events [best papers from 13 of the workshops co-located with the ISWC 2018 conference], vol. 36 of Studies on the Semantic Web, pp. 203–218. IOS Press (2018)
12. Kostopoulos, C., Mouchakis, G., Troumpoukis, A., Prokopaki-Kostopoulou, N., Charalambidis, A., Konstantopoulos, S.: KOBE: Cloud-native open benchmarking engine for federated query processors. In: Verborgh, R., et al. (eds.) ESWC 2021. LNCS, vol. 12731, pp. 664–679. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77385-4_40

13. Montoya, G., Skaf-Molli, H., Molli, P., Vidal, M.-E.: Decomposing federated queries in presence of replicated fragments. J. Web Semant. **42**, 1–18 (2017)
14. Montoya, G., Vidal, M.-E., Corcho, O., Ruckhaus, E., Buil-Aranda, C.: Benchmarking federated SPARQL query engines: are existing testbeds enough? In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012. LNCS, vol. 7650, pp. 313–324. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35173-0_21
15. Oguz, D., Ergenc, B., Yin, S., Dikenelli, O., Hameurlain, A.: Federated query processing on linked data: a qualitative survey and open challenges. Knowl. Eng. Rev. **30**(5), 545–563 (2015)
16. Quilitz, B., Leser, U.: Querying distributed RDF data sources with SPARQL. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 524–538. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68234-9_39
17. Rakhmawati, N.A., Saleem, M., Lalithsena, S., Decker, S.: Qfed: query set for federated SPARQL query benchmark. In: Indrawan-Santiago, M., Steinbauer, M., Nguyen, H.-Q., Min Tjoa, A., Khalil, I., Anderst-Kotsis, G. (eds.) Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services, Hanoi, 4–6 December 2014, pp. 207–211. ACM (2014)
18. Saleem, M., Hasnain, A., Ngonga Ngomo, A.-C.: Largerdfbench: a billion triples benchmark for SPARQL endpoint federation. J. Web Semant. **48**, 85–125 (2018)
19. Saleem, M., Khan, Y., Hasnain, A., Ermilov, I., Ngonga Ngomo, A.-C.: A fine-grained evaluation of SPARQL endpoint federation systems. Semant. Web **7**(5), 493–518 (2016)
20. Saleem, M., Potocki, A., Soru, T., Hartig, O., Ngonga Ngomo, A.-C.: Costfed: cost-based query optimization for sparql endpoint federation. In: 14th International Conference on Semantic Systems (SEMANTICS), pp. 163–174. Elsevier (2018)
21. Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: FedBench: a benchmark suite for federated semantic data query processing. In: Aroyo, L., et al. (eds.) ISWC 2011. LNCS, vol. 7031, pp. 585–600. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25073-6_37
22. Schmidt, M., Hornung, T., Meier, M., Pinkel, C., Lausen, G.: SP2Bench: a SPARQL performance benchmark. In: de Virgilio, R., Giunchiglia, F., Tanca, L. (eds.) Semantic Web Information Management: A Model-Based Perspective, pp. 371–393. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-04329-1_16
23. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: a federation layer for distributed query processing on linked open data. In: Antoniou, G., et al. (eds.) The Semanic Web: Research and Applications, pp. 481–486. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21064-8_39
24. Wang, X., Tiropanis, T., Davis, H.C.: LHD: optimising linked data query processing using parallelisation. In: Bizer, C., Heath, T., Berners-Lee, T., Hausenblas, M., Auer, S. (eds.) Proceedings of the WWW2013 Workshop on Linked Data on the Web, Rio de Janeiro, 14 May, 2013, vol. 996 of CEUR Workshop Proceedings. CEUR-WS.org (2013)