μ-Bench: Real-world Micro Benchmarking for SPARQL Query Processing over Knowledge Graphs

Muhammad Saleem
saleem@mail.uni-paderborn.de
The Data Science (DICE) group
University of Paderborn, Germany
Germany

Adnan Akhter
akhter@informatik.uni-leipzig.de
The Data Science (DICE) group
University of Paderborn, Germany
Germany

Sahar Vahdati
vahdati@infai.org
Institute for Applied Informatics (InfAI)
Dresden,
Germany

Axel Cyrille Ngonga Ngomo
axel.ngonga@upb.de
The Data Science (DICE) group
University of Paderborn, Germany
Germany

ABSTRACT

Real-world SPARQL querying benchmarks, which make use of the real-world RDF datasets and/or SPARQL queries, are the key element in testing the performance of different RDF Knowledge graph management systems in real-world settings. Over the last years, various real-world SPARQL querying benchmarks have been proposed. Although useful for general purpose SPARQL benchmarking, they do not allow generating microbenchmarks, i.e., small customized benchmarks according to the user specified criteria for a specific use case. These microbenchmarks are important to perform component-based testing, hence pinpoint pros and cons of the systems at micro level. We propose μ-Bench, a microbenchmarking framework for SPARQL query processing over RDF knowledge graphs. The framework makes use of the real-world (collected from query logs of public SPARQL endpoints) SPARQL queries to generate customized benchmarks according to the user defined criteria. The framework utilizes various clustering algorithms to select diverse benchmarks from the given input query log. We generated various microbenchmarks and evaluated state-of-the-art knowledge graph engines. The evaluation results show that specialized microbenchmarking is crucial for identifying the limitations of the various SPARQL query processing engines and other corresponding components.

CCS CONCEPTS

• **Information systems** → *Presentation of retrieval results.*

ACM Reference Format:
Muhammad Saleem, Adnan Akhter, Sahar Vahdati, and Axel Cyrille Ngonga Ngomo. 2022. μ-Bench: Real-world Micro Benchmarking for SPARQL Query Processing over Knowledge Graphs. In *Proceedings of International Joint Conference on Knowledge Graphs (IJCKG '2022).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

IJCKG '2022, October 27–29, 2022, Hangzhou, China

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Over the last few years, an enormous number of RDF datasets have been published in the form of knowledge graphs. Several Big RDF datasets such as UniProt[1] (105.5 billion triples), Linked TCGA [18] (20.4 billion triples), Linked Geo Data [5] (20 billion triples) etc. have been added into the Linking Open Data (LOD) cloud. Storing and querying such massive knowledge graphs still remains a challenging task. To this end, various triplestores (RDF knowledge graph storage and querying engines) have been developed for the last two decades [1]. The performance of these triplestores is the central importance for the applications based on RDF knowledge graphs. Consequently, several RDF benchmarks have been proposed that allow to assess the performance of such triplestores [7, 23].

Although many of thee existing benchmarks [2, 8, 10, 12, 16, 23–27] are useful for general purpose SPARQL benchmarking, they do not allow generating customized benchmarks according to the user specified criteria for a specific use case. For example, a user may be interested to test a given a triplestore based on real-world data and SPARQL queries with the following properties: (1) the number of triple patterns in the SPARQL queries should be at least three with at least one join variable, (2) the result size of the query should be greater than 1000, and (3) the number of projection variables in the SPARQL queries should be exactly two. Such use-case specific micro benchmarks are important to perform components-based testing and hence pinpoint pros and cons of the systems at micro level. The FEASIBLE [19] benchmark aims to generate customized benchmarks. However, their approach make use of a single clustering method. Therefore, the variety of the benchmarks generated by this framework is limited. In addition, it requires a pre-processing step to clean the given input queries and convert it into an accepted format by FEASIBLE , which is time consuming.

Previous works [2, 11, 23] pointed out various datasets and query features that can affect the performance of triplestores. For example, the benchmarks analysis [23] pointed out that the number of projection variables, join vertices, triple patterns, the result sizes, and the join vertex degree are the top five SPARQL features that have the highest impact on the overall query execution time. A given SPARQL benchmark should have sufficient diversity in these

[1] UniProt: https://www.uniprot.org/

features among the set of queries included in the benchmark [2, 23]. Testing triplestores with less diverse SPARQL benchmarks may favour a particular type of triplestores and hence biased the overall performance evaluation [2, 23].

We propose μ-Bench, a SPARQL benchmarking framework that allows users to generate customized microbenchmarks on various SPARQL features. The framework requires a set of SPARQL queries (real or synthetic) to be provided as an input. The selection of the microbenchmarks (subset of the input queries) is performed by using state-of-the-art distance-based clustering methods. The user is able to apply various filters on the given SPARQL features to be considered while generating microbenchmarks. The framework can be employed both for real-world and synthetic microbenchmarking. In this paper, we mostly focus on real-world benchmarking[2]. The required input queries can be directly provided from the Linked SPARQL Queries (LSQ) [25] datasets, a set of RDF datasets containing real-world SPARQL queries collected from public SPARQL endpoints. At present, the LSQ datasets describes 43.95 million executions of 11.56 million unique SPARQL queries extracted from the logs of 27 different endpoints.

Our main contributions are as follows:

- The proposed framework provides customized microbenchmarks according to user-defined criteria on important SPARQL features. The framework gives flexibility to perform microbenchmarking based on various clustering methods.
- The framework is directly compatible with LSQ datasets. Given a url for any of the 27 LSQ endpoints, it generates real-world microbenchmarks for 27 different knowledge graphs with a total of 11.56 million queries without any pre-processing of data.
- We evaluate the performance of the different triplestores based on the microbenchmarks generated by our framework. In addition, the supported clustering methods have been compared in terms of diversity of the benchmarks generated by these clustering methods.

μ-Bench is freely available from https://github.com/dice-group/mu-Bench.

2 PRELIMINARIES

In general, a querying benchmark consists of four main components: (1) a set of dataset(s), (2) a set of queries to be executed against the datasets, (3) a set of performance metrics to compare querying engines, and (4) a set of execution rules, e.g, parameters settings. A querying benchmark should carefully select the first two components in order to have sufficient feature variety with respect to various dataset and queries [23].

For microbenchmarking, the literature about SPARQL querying benchmarks [2, 13, 17, 19, 22, 23] highlighted various SPARQL features to be considered while designing querying workloads. Saleem et al. [23] calculated the correlation of various SPARQL features with the query runtime based on different SPARQL query processing engines. The top 10 SPARQL query features with highest impact on the query runtime are reported as: (1) number of projection variables used in the given SPARQL query, (2) the number of join vertices, (3) the number of triple patterns, (4) the query result size, (5) join vertex degree, (6) join-restricted triple pattern selectivity, (7) triple pattern selectivity, (8) number of Basic Graph Patterns (BGPs), (9) number of LSQ features, and (10) BGP-restricted triple pattern selectivity. The study also mentioned the highly used SPARQL clauses (e.g., `LIMIT`, `OPTIONAL`, `ORDER BY`, `DISTINCT`, `UNION`, `FILTER`, `REGEX`) which have direct impact on the runtime performance of triplestores. Our proposed microbenchmarking framework makes use of all of these features while selecting customized use-case specific benchmarks. We now formally provide the definitions for these features which are mostly reused from the previous works [2, 4, 23].

Definition 1 (RDF Term, RDF Triple and RDF Dataset)**.** Assume there are pairwise disjoint infinite sets I, B, and L (IRIs, Blank nodes, and Literals, respectively). Then the RDF term is noted by RT where $RT = I \cup B \cup L$. Any triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an *RDF triple*, where s is called the *subject*, p the *predicate* and o the *object*. An *RDF data set* or data source D is a set of RDF triples $D = \{(s_1, p_1, o_1), \ldots, (s_n, p_n, o_n)\}$.

Definition 2 (Query Triple Pattern and Basic Graph Pattern)**.** By using Definition 1 and assuming an infinite set V of variables, a tuple $\text{tp} \in (I \cup L \cup V \cup B) \times (I \cup V) \times (I \cup L \cup V \cup B)$ is a triple pattern. A Basic Graph Pattern (BGP) is a finite set of triple patterns.

Definition 3 (Basic Graph Pattern)**.** The syntax of a SPARQL Basic Graph Pattern *BGP* expression is defined recursively as follows:

(1) A tuple from $(I \cup L \cup V \cup B) \times (I \cup V) \times (I \cup L \cup V \cup B)$ is a graph pattern (a *triple pattern*).
(2) The expressions (P_1 AND P_2), (P_1 OPTIONAL P_2) and (P_1 UNION P_2) are graph patterns, if bith of the P_1 and P_2 are graph patterns.
(3) The expression (P FILTER R) is a graph pattern, if P is a graph pattern and R is a SPARQL constraint or filter expression.

In our approach, the BGPs are represented as *directed hypergraph* (DH) [21], in which each subject, predicate, and object of a triple pattern represent a single vertex, connected by hyperedges. It is unlike a common SPARQL representation where the subject and object of the triple pattern are connected by an edge, labeled with a predicate name. In this representation, every hyperedge captures a triple pattern such that the subject of the triple pattern becomes the source vertex of a hyperedge and the predicate and object (both combined) of the triple pattern become the target vertices. An example DH representation of a SPARQL query is shown in Figure 1. Unlike common triple pattern representation, the DH representation contains nodes for all three components of the triple patterns. The advantage of this representation is that we can capture predicate-predicate joins, i.e, joins between predicates of the triple patterns of SPARQL query. Formally, our hypergraph representation is defined as follows [21]:

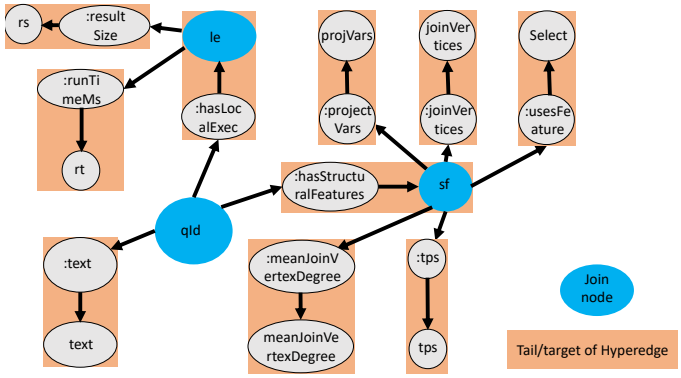
Definition 4 (Directed hypergraph of a BGP)**.** The hypergraph representation of a BGP B is a directed hypergraph $HG = (V, E)$ whose vertices are all the components of all triple patterns in B, i.e., $V = \bigcup_{(s,p,o) \in B} \{s, p, o\}$, and that contains a hyperedge $(S, T) \in E$ for every triple pattern $(s, p, o) \in B$ such that $S = \{s\}$ and $T = (p, o)$.

The representation of a complete SPARQL query as a DH is the union of the representations of the query's BGPs.

[2]We have more real-world queries from public SPARQL endpoints available.

```
Prefix lsq: <http://lsq.aksw.org/vocab#>
SELECT DISTINCT ?qId ?projVars ?joinVertices
?tps ?rs ?rt ?meanJoinVertexDegree {
?qId lsq:text ?text.
?qId lsq:hasStructuralFeatures ?sf .
?sf lsq:projectVars ?projVars .
?sf lsq:joinVertices ?joinVertices .
?sf lsq:tps ?tps .
?sf lsq:meanJoinVertexDegree
        ?meanJoinVertexDegree .
?qId lsq:hasLocalExec ?le .
?le lsq:resultSize ?rs .
?le lsq:runTimeMs ?rt.
Filter (?rt > 50 && ?rs < 50 && ?tps < 10
&& (?projVars = 3 || ?joinVertices >2))}
LIMIT 1000
```

(a) SPARQL query



(b) DH representation

Figure 1: DH representation of the SPARQL query given in Figure 1a. Prefixes are ignored for succinctness

Based on the DH representation of SPARQL queries, we can define the following features of SPARQL queries, which we will be using in μ-Bench :

Definition 5 (Join Vertex)**.** For every vertex $v \in V$ in such a hypergraph, we write $E_{\text{in}}(v)$ and $E_{\text{out}}(v)$ to denote the set of incoming and outgoing edges, respectively; i.e., $E_{\text{in}}(v) = \{(S,T) \in E \mid v \in T\}$ and $E_{\text{out}}(v) = \{(S,T) \in E \mid v \in S\}$. If $|E_{\text{in}}(v)| + |E_{\text{out}}(v)| > 1$, we call v a *join vertex*.

In Figure 1b, all vertexes highlighted in blue are join nodes.

Definition 6 (Join Vertex Degree)**.** Based on the DH representation of the queries the join vertex degree of a vertex v is $\mathcal{JVD}(v) = |E_{in}(v)| + |E_{out}(v)|$, where $E_{in}(v)$ resp. $E_{out}(v)$ is the set of incoming resp. outgoing edges of v.

In Figure 1b, the join vertex degree of the vertex `qId` is three, since it is 3 outgoing hyperedges and no incoming hyperedge.

Definition 7 (Join Vertex Types)**.** A vertex $v \in V$ can be of type *star*, *path*, *hybrid*, or *sink* if this vertex participates in at least one join. A *star* vertex has more than one outgoing edge and no incoming edges. 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*.

In Figure 1b, all the join vertices highlighted in blue are of type `star`, since they all have more than one outgoing hyperedge and no incoming hyperedge .

Definition 8 (Triple Pattern Selectivity)**.** Let tp_i be a triple pattern of a SPARQL query Q and D be a dataset. Furthermore, let N be the total number of triples in D and $Card(tp_i, D)$ be the cardinality of tp_i w.r.t. D, i.e., total number of triples in D that matches tp_i, then the selectivity of tp_i w.r.t. D denoted by $Sel(tp_i, D) = Card(tp_i, D)/N$.

Definition 9 (BGP-Restricted Triple Pattern Selectivity)**.** Consider a Basic Graph Pattern BGP and a triple pattern tp_i belonging to BGP, let $R(tp_i, D)$ be the set of distinct solution mappings (i.e., result set) of executing tp_i over dataset D and $R(BGP, D)$ be the set of distinct solution mappings of executing BGP over dataset D. Then the BGP-restricted triple pattern selectivity denoted by $Sel_{BGP\text{-}Restricted}(tp_i, D)$ is the fraction of distinct solution mappings in $R(tp_i, D)$ that are compatible (as per standard SPARQL semantics [3]) with a solution mapping in $R(BGP, D)$ [2]. Formally, if Ω and Ω' denote the sets underlying the (bag) query results $R(tp_i, D)$ and $R(BGP, D)$, respectively, then

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

Definition 10 (Join-Restricted Triple Pattern Selectivity)**.** Considering a join vertex x in the DH representation of a BGP, let BGP' belonging to BGP be the set of triple patterns that are incidents to x. Furthermore, let tp_i belonging to BGP' be a triple pattern and $R(tp_i, D)$ be the set of distinct solution mappings of executing tp_i over dataset D and $R(BGP', D)$ be the set of distinct solution mappings of executing BGP' over dataset D. Then the *x-restricted* triple pattern selectivity denoted by $Sel_{\mathcal{J}Vx\text{-}Restricted}(tp_i, D)$, is the fraction of distinct solution mappings in $R(tp_i, D)$ that are compatible with a solution mapping in $R(BGP', D)$ [2]. Formally, if Ω and Ω' denote the sets underlying the (bag) query results $R(tp_i, D)$ and $R(BGP', D)$, respectively, then

$$Sel_{\mathcal{J}Vx\text{-}R.}(tp_i, D) = \frac{|\{\mu \in \Omega \mid \exists \mu' \in \Omega' : \mu \text{ and } \mu' \text{ are compatible}\}|}{|\Omega|}$$

Finally, we combine all these important query features into a single composite metric called the *Diversity Score* of the benchmark queries.

Definition 11 (Queries Diversity Score)**.** Let μ_i be the mean and σ_i the standard deviation of a given distribution w.r.t. the i^{th} feature of the said distribution. The overall diversity score DS of the queries is the average coefficient of variation of all the query features k

analyzed in the queries of benchmark B:

$$DS = \frac{1}{k}\sum_{i=1}^{k}\frac{\sigma_i(B)}{\mu_i(B)}$$

Finally, we assume that the reader is familiar with the notion of projection variables.[3]

3 RELATED WORK

The related work in SPARQL querying benchmarking can be broadly divided into two categories namely synthetic data and real-world benchmarks.

3.1 Synthetic SPARQL Benchmarks

The **Train Benchmark** (TrainBench) [26] uses a data generator that produces railway networks in increasing sizes and serializes them in different formats, including RDF. The benchmark contains a total of 16 SPARQL queries. A deeper analysis [23] of the benchmarks shows that the queries included in this benchmark are missing important SPARQL clauses such as `DISTINCT`, `REGEX`, `OPTIONAL`, `UNION`, `LIMIT`, and `ORDERBY`. In addition, the diversity score of the number of joins, join vertex degree, number of triple patterns, the queries result sizes, triple patterns selectivities, join and BGP-restricted triple patterns selectivities, number of BGPs, and queries runtime are below the average value (across all benchmarks included in the study [23]).

The **Waterloo SPARQL Diversity Test Suite** (WatDiv) [2] provides a synthetic data generator that produces RDF data with adjustable structuredness value. It makes use of the query templates to generate SPARQL queries that can be used in the performance evaluation. The queries are generated from different query templates. Currently, there are 50 queries templates available, divided into three use-cases[4] namely *Basic Testing*, *Extensions to Basic Testing*, and *Stress Testing*. This benchmark only contains BGP queries and hence missing the aforementioned important SPARQL clauses. In addition, the diversity score of the number of projection variables, number of joins, join vertex degree, number of triple patterns, the queries result sizes, triple patterns selectivities, join-restricted triple patterns selectivities, number of BGPs, and query runtimes are below the average value (across all benchmarks included in the study [23]).

SP2Bench [24] is based on DBLP bibliographic database and mirrors vital characteristics such as power law distributions or Gaussian curves. The benchmark includes a total of 14 queries. It makes use of the majority of the aforementioned SPARQL clauses. However, the diversity score of the number of joins, queries result sizes, the join vertex degree, triple patterns selectivities, join and BGP-restricted triple patterns selectivities, number of BGPs, and query runtimes are below the average value (across all benchmarks included in the study [23])

The **Berlin SPARQL Benchmark** (BSBM) [8] is also based on query templates to generate SPARQL queries for benchmarking. The queries belong to multiple use cases such as explore, update, and business intelligence. It contains a total of 20 query templates. It also makes use of the majority of the aforementioned SPARQL clauses. However, the diversity score of the number of joins, queries result sizes, the join vertex degree, triple patterns selectivities, BGP-restricted triple patterns selectivities, number of BGPs, and query runtimes are below the average value (across all benchmarks included in the study [23]).

The **Bowlogna** [10] benchmark is based on the Bologna process. It contains a total of 16 queries with a particular focus on analytical and trend inquiries. The queries included in this benchmark are missing important `OPTIONAL` and `UNION` clauses. In addition, the diversity score of the number of projection variables, number of joins, join vertex degree, number of triple patterns, the queries result sizes, triple patterns selectivities, join and BGP-restricted triple patterns selectivities, number of BGPs, and queries runtime are below the average value (across all benchmarks included in the study [23]).

The **LDBC Social Network Benchmark** (SNB) is based on social networking data and proposes two different types of queries workloads for the same data: (1) the *Interactive* workload (SNB-INT) queries are based on the person and its neighborhood. The graph data continuously updated [12] in this workload, and (2) the *Business Intelligence* workload (SNB-BI) focuses on the complex queries with aggregates, collecting information from significant portion of the graph. The data in this workload is static without any updates. The SNB benchmark in total contains 21 queries. The diversity score of the number of projection variables, number of joins, join vertex degree, the queries result sizes, triple patterns selectivities, join and BGP-restricted triple patterns selectivities, number of BGPs, and queries runtime are below the average value (across all benchmarks included in the study [23]).

3.2 Triplestore Benchmarks Using Real Data

FEASIBLE [19] is a benchmark generation framework from query logs of SPARQL endpoints. The benchmarking utilizes a distance-based clustering and the notion of most representative query selection from each cluster. FEASIBLE takes various important SPARQL query features (such as result set sizes, total number of query triple patterns, join vertices and mean join vertices degree) into consideration when generating benchmarks. Currently, it only supports Virtuoso query logs that need to be pre-processed, cleaned and stored in a separate file, which is used as input in the next stage. FEASIBLE can generate benchmarks with varying numbers of queries. It makes use of all aforementioned SPARQL clauses. However, the queries result sizes, BGP-restricted triple patterns selectivities, and queries runtime are below the average value (across all benchmarks included in the study [23]). The **DBpedia SPARQL Benchmark** (DBPSB) [15] is also a cluster-based benchmark generation approach that generates benchmark queries from DBpedia query logs. However, the clustering technique used in DBPSB is different from FEASIBLE. It utilizes DBpedia data and real-world queries templates collected from DBpedia public SPARQL endpoint. It contains a total of 25 query templates. All of the query templates contain `DISTINCT` SPARQL clause while missing important `LIMIT` and `ORDER BY` clauses. In addition, the diversity score of the number of projection variables, number of joins, the queries result sizes, join vertex degree, the queries result sizes, triple patterns selectivities, join and BGP-restricted triple patterns selectivities, and

[3] See https://www.w3.org/TR/sparql11-query/#modProjection.
[4] WatDiv Tests: https://dsg.uwaterloo.ca/watdiv/#tests

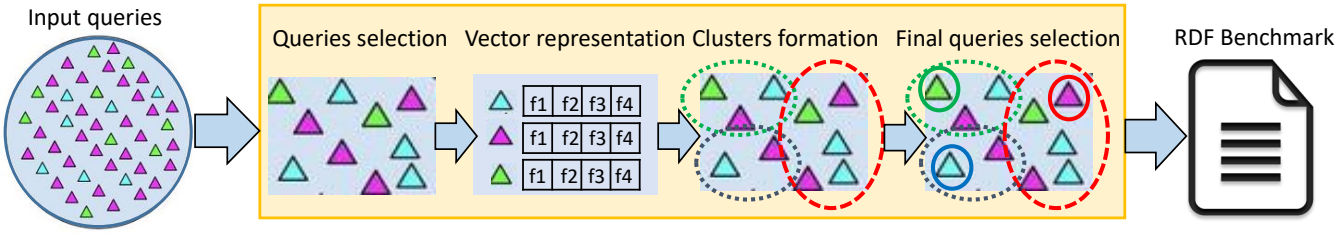


Figure 2: Basic Architecture of the μ-Bench

queries runtime are below the average value (across all benchmarks included in the study [23]).

FishMark [6] is based on FishBase[5] data which is provided both in RDF and SQL versions. The queries are collected from the public web-based application developed for FishBase. It contains a total of 22 queries, which misses important SPARQL clauses such as `DISTINCT`, `LIMIT`, `FILTER`, `REGEX` and `ORDER BY`. In addition, the diversity score of the number of joins, the queries result sizes, join vertex degree, the queries result sizes, triple patterns selectivities, join and BGP-restricted triple patterns selectivities, and queries runtime are below the average value (across all benchmarks included in the study [23]).

BioBench [28] includes five, i.e., Cell, Allie, PDBJ, DDBJ, and UniProt real-world biological datasets and their corresponding queries in their benchmark. It makes use of all of the aforementioned SPARQL clauses. However, the diversity score of the number of joins, the queries result sizes, join vertex degree, the queries result sizes, triple patterns selectivities, join and BGP-restricted triple patterns selectivities, and queries runtime are below the average value (across all benchmarks included in the study [23]).

4 THE μ-BENCH APPROACH

Our benchmark creation problem is defined as follows:

Definition 12 (Benchmark Creation Problem)**.** Let L represent the set of input queries. Our target is to obtain N queries that are not only the best representatives of L but also contain more diverse query features, with $N << |L|$.

For benchmark creation, a set of input SPARQL queries, the required number N of queries and the selection criteria for microbenchmarking are provided by the user.

Figure 2 shows the general architecture of the proposed microbenchmarking framework dubbed μ-Bench, which includes four key components:(1) queries selection, (2) vector representation, (3) clusters formation, and (4) final queries selection. In the subsequent sections, we discuss these steps in detail.

4.1 Queries Selection

The μ-Bench framework requires a set of SPARQL queries (either synthetic or real-worl) to be provided as input. As mentioned before, the framework is directly compatible with the LSQ datasets[6] and user can only provide the LSQ SPARQL endpoint url[7] as input.

The queries selection component selects all those input queries which are according to the selection criteria (i.e., preferences for the personalized benchmark) provided by the user. Our framework completely abides by the semantic web and knowledge graphs principles; the user can specify the selection criteria by using a single SPARQL query. An example of such personalized query is shown in Figure 1a, where the user specified the following criteria for the desired microbenchmark: (1) the result size of the queries included in the benchmark should be greater than 50, (2) the runtimes of all the queries should be less than 50 seconds, (3) the number of triple patterns included in the queries should be less than 10, and (4) the number of projections variables should be either three or the number of join vertices should be greater than two. Please note that beside all the important SPARQL queries features discussed in the previous section, LSQ also provides additional information which can also be used to generate personalized microbenchmarks. We encourage authors to have a look at LSQ schema for the details of the all features included in the LSQ vocabulary[8]. At this stage, a sample of the queries are selected from the given input set of queries. The next step is to convert the selected queries into feature vectors.

4.2 Vector Representation

We represent each SPARQL query as a multi-dimensional vector. The same SPARQL query which was used for query selection in the previous step is also used to get features (which are used in feature vectors) pertaining to each selected query. The list of projection variables represent the features to be used in vectors. For example, the Figure 1a has five projection variables namely ?qID (query id): (1) ?projVars (number of projection variables), (2) ?joinVertices (number of join vertices), (3) ?tps (number of triple patterns), (4) ?rs (result size), and (5) ?meanJoinVertexDegree (mean join vertex degree). In this example, each query will have a corresponding feature vector with size five. The features values are already stored in the LSQ dataset which is provided as input. For example, the SPARQL query given in Figure 1a will have the following features vector [7, 3, 9, 1000, 3], with 7 projection variables, 3 join vertices, 9 triple patterns, assuming the result size is 1000, and average join

[5]FishBase: http://fishbase.org/search.php

[6]We recommend to use it with LSQ datasets, as it provides various information about given SPARQL queries, which can be used to generate personalized benchmarks. It is noteworthy that LSQ framework can be used to RDFize other queries logs or synthetic SPARQL queries as well.

[7]LSQ endpoint: http://lsq.aksw.org/sparql

[8]LSQ vocabulary: https://github.com/AKSW/LSQ/blob/develop/LSQVocab_v2.ttl

vertex degree equals 3. In order to make sure, the features with very high values (e.g. 1000 in our example) do not bias the overall selection of queries, we normalize all the values in a unit hypercube. This normalization is done such that each of the individual values in every feature vector is divided by the overall maximal value (across all the vectors) for that query feature.

4.3 Clusters Formation

Given a set of normalized vectors, the next step is to generate the required N clusters of the selected queries. To this end, we plot all the normalized vectors in the multidimensional space and use well-known distance-based clustering techniques to generate the required number of N clusters. Currently, our framework supports several distance-based clustering namely FEASIBLE [20], FEASIBLE Exemplars [20], KMeans++, DBSCAN+KMeans++ (Combination of DBSCAN and KMeans where DBSCAN remove outliers while KMeans generate the required number of clusters), and Random selection. Our framework is flexible enough to add more distance-based clustering methods, which allow us to generate the required N number of clusters. We encourage readers to have a look at the corresponding papers for the details of these clustering methods.

4.4 Final Queries Selection

The last step is to only select a single most representative query from each cluster of queries. This is done as follows: for each cluster, first we find the centroid which is the average of the feature vectors of all queries included in the given cluster. Once the centroid is determined, we calculate the distance between each query included in the given cluster to this centroid; a query that has the shortest distance to the centroid is selected as the most representative query of this cluster. After the selection of representative queries, they are stored as an RDF file as the final benchmark output along with a list of features associated with each query. This RDF output can be queried directly using a SPARQL query.

5 EVALUATION AND RESULTS

In this section, we discuss the evaluation setup and evaluation results. We performed two types of experiments: (1) evaluation of the diversity scores and the benchmark generation times of the supported clustering methods, (2) performance evaluation of the well-known triplestores using microbencharks generated by our framework. The goal of the first experiment was to test the scalability of the supported clustering methods by our framework and to determine which of them generates the most diverse benchmark. The purpose of the second experiment was not to determine the fastest state-of-the-art triplestore, rather we want to show that microbenchmarking can be helpful to perform more fine-grained evaluation and hence pinpoint new insights.

5.1 Experimental Setup

Datasets In our evaluation, we want to perform experiments on real-world datasets and queries. To this end, we selected four different real-world dataset namely DBpedia[9], Side Effect Resource (Sider[10]), Semantic Web Dog Food (SWDF[11]) and Linked Geo Data (LGD[12]). We chose these datasets because: (1) they are from diverse domains, (2) their real-world SPARQL queries are available from LSQ, (3) they have varying structuredness and sizes.

Microbenchmarks For each of the above four datasets, we generate 6 different microbenchmarks with varying query sizes, i.e., 10-queries, 25-queries, 50-queries, 100-queries, 500-queries and 1000-queries. We carefully chose these values to better show the difference between the diversity scores of the supported clustering methods by our framework.

Hardware and Software Configurations All experiments were run on an Ubuntu-based machine with intel Xeon 2.10 GHz, 64 cores and 512GB of RAM. For triplestores evaluation, we used IGUANA [9], which is a novel SPARQL benchmark execution framework.

Triplestores We compared the performance of four – BlazeGraph, Fuseki, GraphDB and Virtuoso – well-known triplestores using the aforementioned microbenchmarks generated for four different datasets. We chose these triplestores because they are sufficiently scalable to the selected datasets, provide full support for SPARQL 1.1, easy to configure, allow publishing their comparison results, free to use, and used in previous evaluations [2, 14, 20]. We encourage readers to have a look [1] for the details of existing triplestores pertaining to storage and querying functionalities.

Performance Measures: We use the diversity score to compare the diversity of the supported clustering techniques by our framework. In addition, we used the benchmark generation time to compare the scalability of the supported clustering techniques for the problem in hand. To measure the performance of triple stores, we use the standard Queries per Second (QpS).

5.2 Results

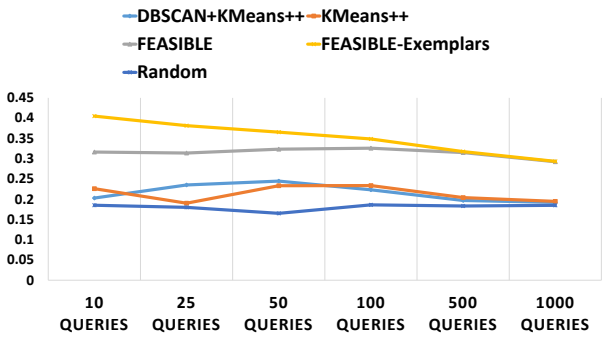
Diversity Score. Figure 3a to Figure 3d show the overall diversity score of all six microbenchmarks for each of the datasets. As an overall comparison, it has been observed that FEASIBLE-Exemplars generates the most diverse benchmarks, followed by FEASIBLE, KMeans++, DBSCAN+KMeans++, Random, respectively. The reason for FEASIBLE-Exemplars high diversity is due to its clustering method, which is based on selecting exemplars by using longest distances from each other. Thus, features-wise, queries selected by this clustering method are the most diverse from each other. On the other hand, FEASIBLE and KMeans++ are centroid-based: they calculate centroids and select samples, instead of selecting samples based on the longest distances. The diversity score of DBSCAN+KMeans++ is lower than KMeans++. This is because the removal of outliers (the most diverse queries) by DBSCAN reduced the overall diversity score. The diversity of the random selection is the lowest because it does not follow a particular method for the selection of queries. Finally, we can see a general trend in decreasing the diversity score with the increasing number of queries included in the benchmark. This is because with the increase in number of queries, the standard deviation among the query features is decreased and hence the overall diversity score is decreased.

[9] https://www.dbpedia.org/

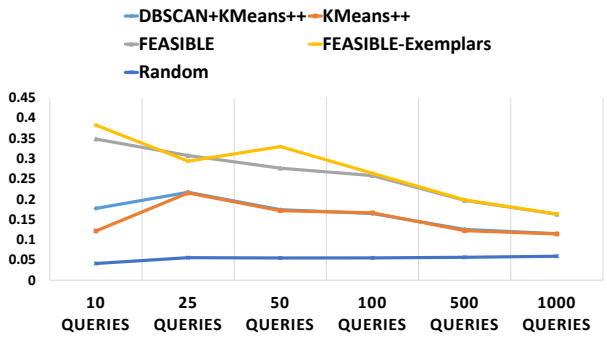
[10] http://sideeffects.embl.de/

[11] http://www.scholarlydata.org/

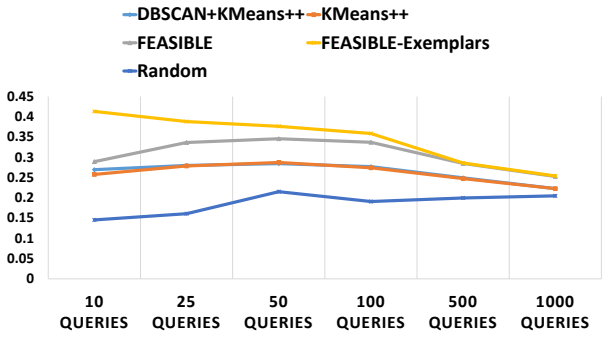
[12] http://linkedgeodata.org/



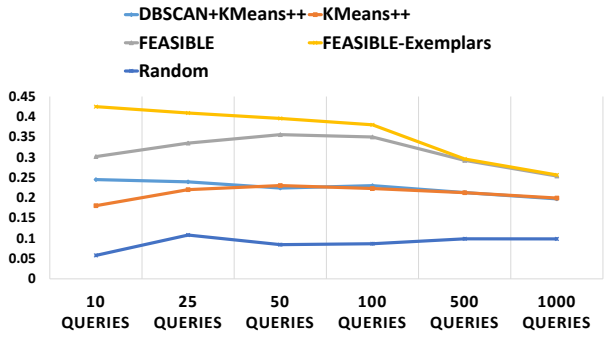
(a) SWDF Benchmarks Diversity Score



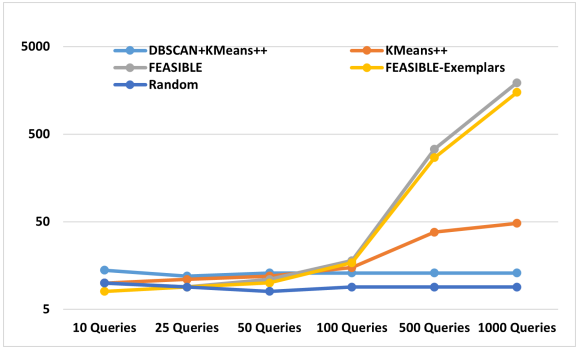
(b) DBpedia Benchmarks Diversity Score



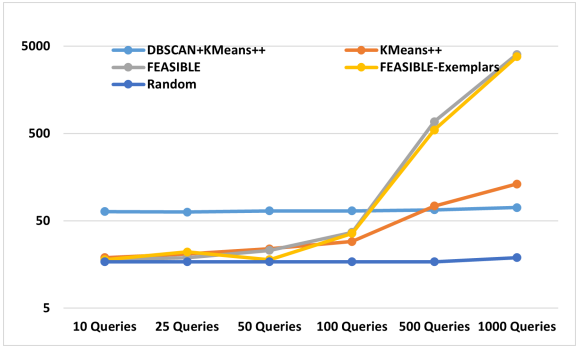
(c) Sider Benchmarks Diversity Score



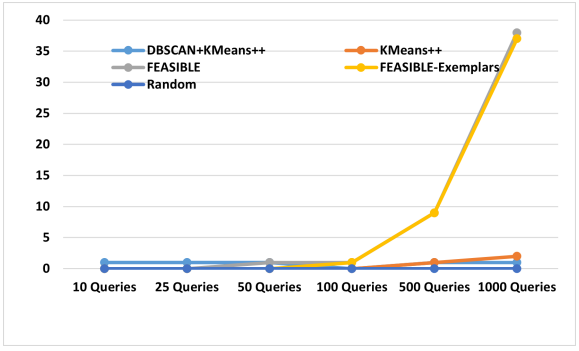
(d) LinkedGeoData Benchmarks Diversity Score



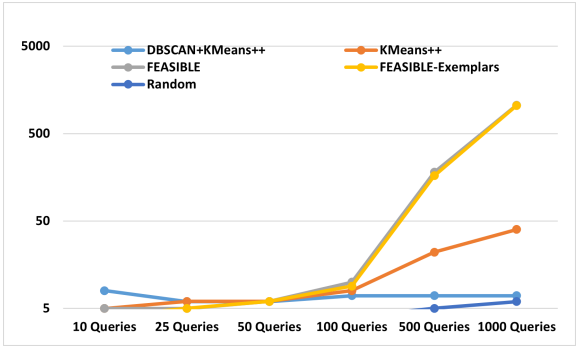
(e) SWDF Benchmarks Generation Time



(f) DBpedia Benchmarks Generation Time

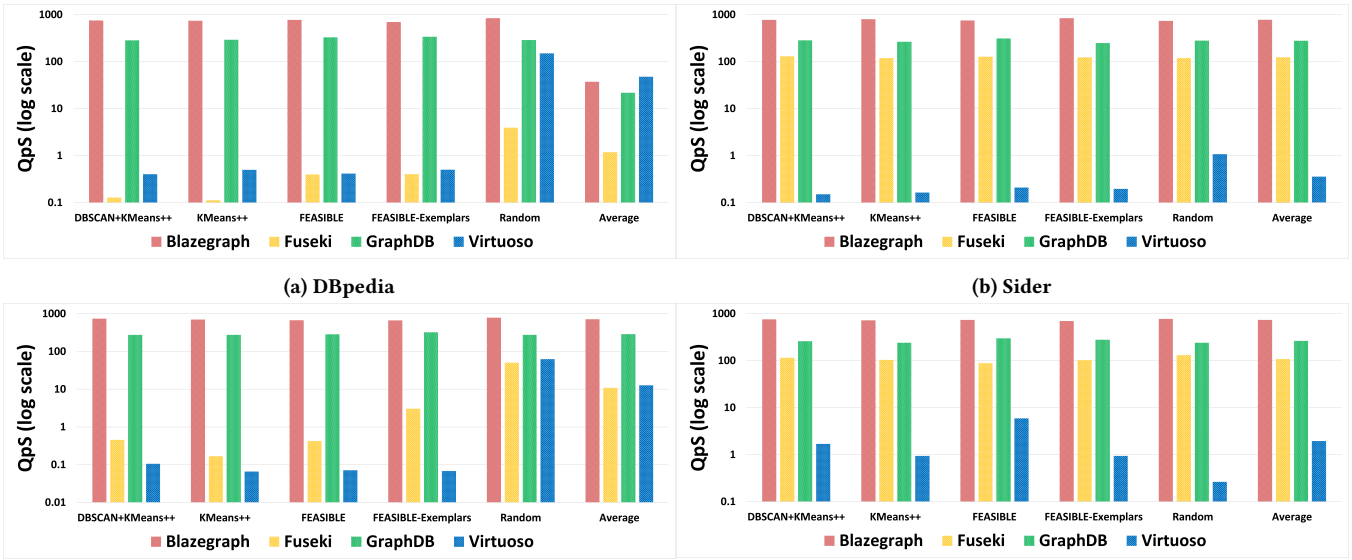


(g) Sider Benchmarks Generation Time



(h) LinkedGeoData Benchmarks Generation Time

Figure 3: Benchmarks Diversity Scores and generation time (seconds) of all four datasets



(a) DBpedia

(b) Sider

(c) SWDF

(d) LinkedGeoData

Figure 4: Queries per Second (QpS) for the selected triplestores and datasets

Benchmark Generation Time. Figure 4a to Figure 4d show the total time (in seconds) taken by selected clustering techniques to generate all six benchmarks for each of the selected datasets. We can clearly observe that FEASIBLE clustering takes the highest time to generate benchmarks, followed by FEASIBLE-Exemplars, KMeans++, DBSCAN+KMeans++ and Random. The reason for this is due to the time complexity of these algorithms. The time complexity of FEASIBLE is $O(d|L|N)$, where |L|is total number of input queries, N is the required number of queries (equal to the number of exemplars), and d is vector dimension (In our experiments, d was 10). The time complexity of FEASIBLE-Exemplars is slightly less than FEASIBLE. This is because (same like FEASIBLE) it only calculates the exemplars, but does not perform the additional step to select the most representative queries. The time complexity of KMeans++ supported by our approach is $O(t * N * |L|)$, where t is the number of iterations (set to 7 in our experiments). The time taken by DBSCAN+KMeans++ is less than KMeans++ because the DBSCAN clustering removed many outliers and hence the total number of input queries (i.e., N) were reduced. Finally, we can observe that although both FEASIBLE and FEASIBLE-exemplars generate the most diverse benchmarks, they might not be very scalable to hundreds of millions of input queries. In our experiments, we obtained the number of input queries from LSQ endpoint as follows: 4,258,941 queries from DBpedia, 277,766 from SWDF, 61,897 queries from LGD, and 47,423 from Sider. The benchmark generation time also reflects that the highest time is needed for DBpedia, followed by SDF, LGD, and Sider, respectively.

Performance of Triplestores. To measure the performance of triplestores, we used 1000 queries microbenchmark from DBpedia, 500 queries microbenchmark from Sider, 100 queries microbenchmark SWDF and 50 queries microbenchmark from LinkedGeoData. Figure 4 shows the evaluation results for these microbenchmarks. For DBpedia, Virtuoso is the fastest triplestore, followed by Blazegraph, GraphDB, and Fuseki, respectively. For Sider, Blazegraph is the fastest triplestore, followed by GraphDB, Fuseki, and Virtuoso, respectively. For SWD, Blazegraph is the fastest triplestore, followed by GraphDB, Virtuoso, and FUSEKI, respectively. Finally, for LGD, Blazegraph is the fastest triplestore, followed by GraphDB, Fuseki, and Virtuoso, respectively. The results clearly suggest that the ranking of the triplestores can be changed by using different microbenchmarks from different datasets.

6 CONCLUSION

In this paper, we presented a microbenchmarking framework for RDF knowledge graphs. The framework is directly compatible with LSQ datasets and hence users need not to prepare the set of input queries along with the required query features. The framework provides various clustering methods to generate microbenchmarks. The evaluation result suggests that FEASIBLE-exemplars generates the most diverse benchmarks. However, it may need more powerful hardware to generate microbenchmarks when the input queries are possibly in billions. The evaluation result also suggested that the performance ranking of the triplestores changes with different microbenchmarks, hence microbenchmarking could be helpful to perform a fine-grained evaluation of triplestores.

In the future, we want to perform more extensive experiments to test the scalability of the proposed approach.

ACKNOWLEDGMENTS

This work was partially supported by the German Federal Ministry of Education and Research (BMBF) within the EuroStars project E!114681 3DFed under the grant no 01QE2114, project RAKI (01MD19012D) and project KnowGraphs (No 860801).

REFERENCES

[1] Waqas Ali, Muhammad Saleem, Bin Yao, Aidan Hogan, and Axel-Cyrille Ngonga Ngomo. 2021. A survey of RDF stores & SPARQL engines for querying knowledge graphs. *The VLDB Journal* (2021), 1–26.

[2] Günes Aluç, Olaf Hartig, M. Tamer Özsu, and Khuzaima Daudjee. 2014. Diversified Stress Testing of RDF Data Management Systems. In *ISWC*. 197–212. https://doi.org/10.1007/978-3-319-11964-9_13

[3] Marcelo Arenas, Claudio Gutiérrez, and Jorge Pérez. 2009. On the Semantics of SPARQL. In *Semantic Web Information Management - A Model-Based Perspective*. Springer, 281–307. https://doi.org/10.1007/978-3-642-04329-1_13

[4] Marcelo Arenas and Jorge Pérez. 2012. *Federation and Navigation in SPARQL 1.1*. Springer.

[5] Sören Auer, Jens Lehmann, and Sebastian Hellmann. 2009. Linkedgeodata: Adding a spatial dimension to the web of data. In *International Semantic Web Conference*. Springer, 731–746.

[6] Samantha Bail, Sandra Alkiviadous, Bijan Parsia, David Workman, Mark van Harmelen, Rafael S. Gonçalves, and Cristina Garilao. 2012. FishMark: A Linked Data Application Benchmark. In *Proceedings of the Joint Workshop on Scalable and High-Performance Semantic Web Systems*. 1–15. http://ceur-ws.org/Vol-943/SSWS_HPCSW2012_paper1.pdf

[7] Luigi Bellomarini, Emanuel Sallinger, and Sahar Vahdati. 2020. Knowledge graphs: the layered perspective. In *Knowledge Graphs and Big Data Processing*. Springer, Cham, 20–34.

[8] Christian Bizer and Andreas Schultz. 2009. The Berlin SPARQL Benchmark. *Int. J. Semantic Web Inf. Syst.* 5, 2 (2009), 1–24. https://doi.org/10.4018/jswis.2009040101

[9] Felix Conrads, Jens Lehmann, Muhammad Saleem, Mohamed Morsey, and Axel-Cyrille Ngonga Ngomo. 2017. IGUANA: A Generic Framework for Benchmarking the Read-Write Performance of Triple Stores. In *ISWC*. Springer, 48–65. https://doi.org/10.1007/978-3-319-68204-4_5

[10] Gianluca Demartini, Iliya Enchev, Marcin Wylot, Joël Gapany, and Philippe Cudré-Mauroux. 2011. BowlognaBench - Benchmarking RDF Analytics. In *Data-Driven Process Discovery and Analysis SIMPDA*. Springer, 82–102. https://doi.org/10.1007/978-3-642-34044-4_5

[11] Anastasia Dimou, Sahar Vahdati, Angelo Di Iorio, Christoph Lange, Ruben Verborgh, and Erik Mannens. 2017. Challenges as enablers for high quality Linked Data: insights from the Semantic Publishing Challenge. *PeerJ Computer Science* 3 (2017), e105.

[12] Orri Erling, Alex Averbuch, Josep-Lluis Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat-Pérez, Minh-Duc Pham, and Peter A. Boncz. 2015. The LDBC Social Network Benchmark: Interactive Workload. In *SIGMOD*. ACM, 619–630. https://doi.org/10.1145/2723372.2742786

[13] Olaf Görlitz, Matthias Thimm, and Steffen Staab. 2012. SPLODGE: Systematic Generation of SPARQL Benchmark Queries for Linked Open Data. In *ISWC*. Springer, 116–132. https://doi.org/10.1007/978-3-642-35176-1_8

[14] Manzoor Ali Hashim Khan, Axel-Cyrille Ngonga Ngomo, and Muhammad Saleem. 2021. When is the Peak Performance Reached? An Analysis of RDF Triple Stores. In *Further with Knowledge Graphs: Proceedings of the 17th International Conference on Semantic Systems, 6-9 September 2021, Amsterdam, The Netherlands*, Vol. 53. IOS Press, 154.

[15] Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. 2011. DBpedia SPARQL Benchmark - Performance Assessment with Real Queries on Real Data. In *ISWC*. Springer, 454–469. https://doi.org/10.1007/978-3-642-25073-6_29

[16] Muhammad Saleem, Muhammad Intizar Ali, Aidan Hogan, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. 2015. LSQ: The Linked SPARQL Queries Dataset. In *ISWC*. Springer, 261–269. https://doi.org/10.1007/978-3-319-25010-6_15

[17] Muhammad Saleem, Ali Hasnain, and Axel-Cyrille Ngonga Ngomo. 2018. LargeRDFBench: A billion triples benchmark for SPARQL endpoint federation. *J. Web Sem.* 48 (2018), 85–125. https://doi.org/10.1016/j.websem.2017.12.005

[18] Muhammad Saleem, Maulik R. Kamdar, Aftab Iqbal, Shanmukha Sampath, Helena F. Deus, and Axel-Cyrille Ngonga Ngomo. 2014. Big linked cancer data: Integrating linked TCGA and PubMed. *Journal of Web Semantics* 27-28 (2014), 34–41. https://doi.org/10.1016/j.websem.2014.07.004 Semantic Web Challenge 2013.

[19] Muhammad Saleem, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. 2015. FEASIBLE: a feature-based SPARQL benchmark generation framework. In *ISWC*. Springer, 52–69.

[20] Muhammad Saleem, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. 2015. Feasible: A feature-based sparql benchmark generation framework. In *International Semantic Web Conference*. Springer, 52–69.

[21] Muhammad Saleem, Alexander Potocki, Tommaso Soru, Olaf Hartig, and Axel-Cyrille Ngonga Ngomo. 2018. CostFed: Cost-Based Query Optimization for SPARQL Endpoint Federation. In *SEMANTICS (Procedia Computer Science, Vol. 137)*. Elsevier, 163–174. https://doi.org/10.1016/j.procs.2018.09.016

[22] Muhammad Saleem, Claus Stadler, Qaiser Mehmood, Jens Lehmann, and Axel-Cyrille Ngonga Ngomo. 2017. SQCFramework: SPARQL Query Containment Benchmark Generation Framework. In *K-CAP*. 28:1–28:8. https://doi.org/10.1145/3148011.3148017

[23] Muhammad Saleem, Gábor Szárnyas, Felix Conrads, Syed Ahmad Chan Bukhari, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. 2019. How representative is a sparql benchmark? an analysis of rdf triplestore benchmarks. In *The World Wide Web Conference*. 1623–1633.

[24] Michael Schmidt et al. 2009. SP2Bench: A SPARQL Performance Benchmark. In *Semantic Web Information Management - A Model-Based Perspective*. 371–393. https://doi.org/10.1007/978-3-642-04329-1_16

[25] Claus Stadlera, Muhammad Saleema, Qaiser Mehmoodb, Carlos Buil-Arandac, Michel Dumontierd, Aidan Hogane, and Axel-Cyrille Ngonga Ngomoa. 2022. LSQ 2.0: A Linked Dataset of SPARQL Query Logs. *Semantic Web Journal* (2022).

[26] Gábor Szárnyas, Benedek Izsó, István Ráth, and Dániel Varró. 2018. The Train Benchmark: Cross-technology performance evaluation of continuous model queries. *Softw. Syst. Model.* 17, 4 (2018), 1365–1393. https://doi.org/10.1007/s10270-016-0571-8

[27] Gábor Szárnyas, Arnau Prat-Pérez, Alex Averbuch, József Marton, Marcus Paradies, Moritz Kaufmann, Orri Erling, Peter A. Boncz, Vlad Haprian, and János Benjamin Antal. 2018. An early look at the LDBC Social Network Benchmark's Business Intelligence workload. In *GRADES-NDA at SIGMOD*. ACM, 9:1–9:11. https://doi.org/10.1145/3210259.3210268

[28] Hongyan Wu et al. 2014. BioBenchmark Toyama 2012: An evaluation of the performance of triple stores on biological data. *J. Biomedical Semantics* 5 (2014), 32. https://doi.org/10.1186/2041-1480-5-32