Semantic Web 1 (2019) 1–5
IOS Press

An Empirical Evaluation of Cost-based Federated SPARQL Query Processing Engines

Umair Qudus [a], Muhammad Saleem [b] , Axel-Cyrille Ngonga Ngomo [c] and Young-koo Lee [a,*]
[a] *DKE, Kyung Hee University, South Korea*
E-mail:{umair.qudus,yklee}@khu.ac.kr
[b] *AKSW, Leipzig, Germany*
E-mail: {lastname}@informatik.uni-leipzig.de
[c] *University of Paderborn, Germany*
E-mail: axel.ngonga@upb.de

Abstract.
Finding a good query plan is key to the optimization of query runtime. This holds in particular for cost-based federation engines, which make use of cardinality estimations to achieve this goal. A number of studies compare SPARQL federation engines across different performance metrics, including query runtime, result set completeness and correctness, number of sources selected and number of requests sent. However, although they are informative, these metrics are generic and unable to quantify and evaluate the accuracy of the cardinality estimators of cost-based federation engines. In addition, to thoroughly evaluate cost-based federation engines, the effect of estimated cardinality errors on the overall query runtime performance must be measured. In this paper, we address this challenge by presenting novel evaluation metrics targeted at a fine-grained benchmarking of cost-based federated SPARQL query engines. We evaluate the query planners of five different cost-based federated SPARQL query engines using LargeRDFBench queries. Our results suggest that our metrics are clearly correlated with the overall query runtime performance of the selected federation engines, and can hence provide important solutions when developing the future generations of federation engines.

Keywords: SPARQL, Benchmarking, cost-based, cost-free, federated, Querying

1. Introduction

The availability of increasing amounts of data published in RDF has led to the genesis of many federated SPARQL query engines. These engines vary widely in their approaches to generating a good query plan [1–3]. In general, there exist several possible plans that a federation engine can consider when executing a given query. These plans have a different cost in terms of the resources required and the overall query execution time. Selection of the best possible plan with minimum cost is hence of key importance when devising cost-based federation engines; a fact which is corroborated by a plethora of works in database research [4, 5].

In SPARQL query federation, index-free (heuristics-based) [6, 7] and index-assisted (cost-based) [8–17] engines are most commonly used for federated query processing [1]. The heuristics-based federation engines do not store any pre-computed statistics and hence mostly use different heuristics to optimize their query plans [6]. Cost-based engines make use of an index with pre-computed statistics about the datasets [1]. Using cardinality estimates as principal input, such engines make use of cost models to calculate the cost of different query joins and generate optimized query plans. Most state-of-the-art cost-based federated SPARQL process-

*Corresponding author. E-mail: yklee@khu.ac.kr

ing engines [8, 9, 11–17] achieve the goal of optimizing their query plan by first estimating the cardinality of the query's triple patterns. Then, they use this information to estimate the cardinality of the joins involved in the query. A cost model is then used to compute the cost of performing different query joins while considering network communication costs. One of the query plans with minimum execution costs is finally selected for result retrieval. Since the principle input for cost-based query planning is the cardinality estimates, the accuracy of these estimates is crucial to achieve a good query plan.

The performance of federated SPARQL query processing engines has been evaluated in many recent studies [1, 8, 11, 13, 18–25] by using different federated benchmarks [26–34]. Performance metrics, including query execution time, number of sources selected, source selection time, query planning time, answer completeness and correctness, time for the first answer, and throughput, are usually reported in these studies. Recently, Acosta et al. [24] proposed new metrics to measure the continuous efficiency of query processing approaches. While these metrics allow the evaluation of certain components (e.g., the source selection model), they cannot be used to evaluate the accuracy of the cardinality estimators of the cost-based federation engines. Consequently, they are unable to show how the estimated cardinality errors affect the overall query runtime performance of federation engines.

In this paper, we address the problem of measuring the accuracy of the cardinality estimators of federated SPARQL engines, as well as the effect of these errors on the overall query runtime performance. In particular, we propose metrics[1] for measuring errors in the cardinality estimations of (1) triple patterns, (2) joins between triple patterns, and (3) query plans. We correlate these errors with the overall query runtime performance of state-of-the-art, cost-based SPARQL federation engines. The observed results show that these metrics are significantly correlated with the overall runtime performances. In summary, the contributions of this work are as follows:

- We propose metrics to measure the errors in cardinality estimations of cost-based federated engines. These metrics allow a fine-grained evaluation of cost-based federated SPARQL query engines and uncover novel insights about the performance of these types of federation engines that were not reported in previous works evaluating federated SPARQL engines.
- We measure the correlation of the values of the novel metrics with the overall query runtimes. We show that some of these metrics have a strong correlation with runtimes and can hence be used as predictors for the overall query execution performance.
- We present an empirical evaluation of five—CostFed[8], Odyessey[11], SemaGrow[13], LHD[9] and SPLENDID[12]—state-of-the-art cost-based SPARQL federation engines on LargeRDFBench [26] by using the proposed metrics.

The rest of the paper is organized as follows: In Section 2, we present related work. A motivating example is given in Section 3. In Section 4, we present our novel metrics for the evaluation of cost-based federation engines. The evaluation of these engines with proposed metrics is shown in Section 6. Finally, we conclude in Section 7.

2. Related Work

In this section, we focus on the performance metrics used in the state of the art to compare federated SPARQL query processing engines. Based on the previous federated SPARQL benchmarks [26–28] and performance evaluations [6, 8–13, 18, 19, 22, 24, 25] (see Table 1 for an overview), the performance metrics used for federated SPARQL engines comparison can be categorized as:

- **Index-Related**: Index-assisted approaches [1] make use of stored dataset statistics to generate an optimized query execution plan. The indexes are pre-computed by collecting information from available federated datasets. This is usually a one-time process. However, later updates are required to ensure the result-set completeness of the query processing. The index generation time and its compression ratio (w.r.t. overall dataset size) are important measures to be considered when devising index-assisted federated engines.
- **Query-Processing-Related**: This category contains metrics related to the query processing capabilities of the federated SPARQL engines. The total number of triple-pattern-wise sources selected, number of `ASK` requests used to perform source selection, source selection time, query planning

[1] Our proposed metric is open-source and available online at https://github.com/dice-group/CostBased-FedEval

	Index		Processing					Network		Res		RS		Add	
	Cr	Gt	Qp	#Ts	Qet	#A	Sst	#Tt	#Er	Cu	Mu	Cp	Ct	@T	@K
CostFed[8]	✓	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗
SPLENDID[12]	✗	✗	✗	✓	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
SemaGrow[13]	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Odyssey[11]	✗	✗	✓	✓	✓	✗	✗	✓	✓	✗	✗	✓	✗	✗	✗
LHD[9]	✗	✗	✗	✗	✓	✗	✗	✓	✓	✓	✓	✓	✗	✗	✗
DARQ[10]	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
ANAPSID[22]	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
MULDER[19]	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
FedX[6]	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗
Lusail[18]	✗	✗	✓	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
BioFed[25]	✗	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✗	✗

Table 1: Metrics used in the existing federated SPARQL query processing systems, **Res:** Resource Related, **RS:** Result Set Related, **Add:** Additional, **Cr:** index compression ratio, **Gt:** the index/summary generation time, **Qp:** Query Planning time, **#Ts:** total number of triple pattern-wise sources selected, **Qet:** the average query execution time, **#A:** total number of SPARQL ASK requests submitted, **Sst:** the average source selection time, **#Tt:** number of transferred tuples, **#Er:** number of endpoint requests, **Cu:** CPU usage, **Mu:** Memory usage, **Cp:** Result Set completeness, **Ct:** Result Set correctness, **@K:** dief@k, **@T:** dief@t

time, and overall query runtime are the reported metrics in this category.

- **Network-Related:** Federated engines collect information from multiple data sources, e.g., SPARQL endpoints. Thus, it is important to minimize the network traffic generated by the engines during query processing. The number of transferred tuples and the number of endpoint requests generated by the federation engine are the two network related metrics used in existing federated SPARQL query processing evaluations.
- **Result-Set-Related:** Two systems are only comparable if they produce exactly the same results. Therefore, result set correctness and completeness are the two most important metrics in this category.
- **Resource-Related:** The CPU and memory resources consumed during query processing dictate the query load an engine can bear. Hence, they are of importance when evaluating the performance of federated SPARQL engines.
- **Additional:** Two metrics dief@t and dief@k are proposed to measure continuous efficiency of query processing approaches.

All of these metrics are helpful to evaluate the performance of different components of federated query engines. However, none of these metrics can be used to evaluate the accuracy of the cardinality estimators of cost-based federation engines. Consequently, studying the effect of estimated cardinality errors on the overall query runtime performance of federation engines cannot be conducted based on these metrics. To overcome these limitations, we propose metrics for measuring errors in cardinality estimations of triple patterns, joins between triple patterns, and overall query plan, and show how these metrics are affecting the overall runtime performance of federation engines.

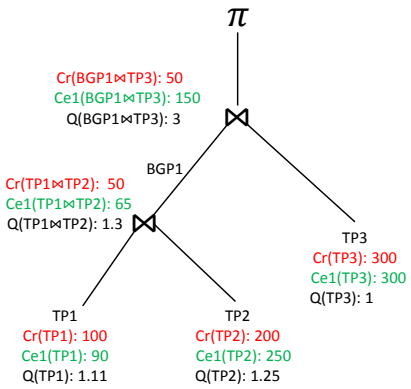
3. Motivating Example

In this section, we present an example to motivate our work and to understand the proposed metrics. We assume that the reader is familiar with the concepts of SPARQL and RDF, including the notions of a triple pattern, the joins between triple patterns, the cardinality (result size) of a triple pattern, and left-deep query execution plans. As aforementioned, most cost-based SPARQL federation engines first estimate individual triple pattern cardinality and use this information to estimate the cardinality of joins found in the query. Finally, the query execution plan is generated by ordering the joins. In general, the optimizer first selects the triple patterns and joins with minimum estimated cardinalities [8].

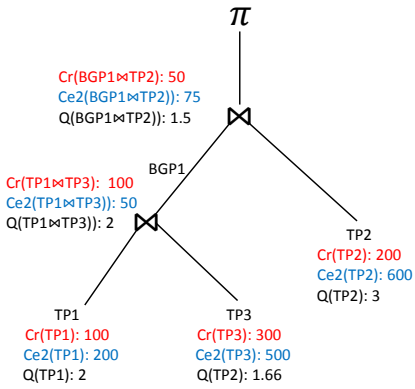
Figure 1 shows a motivating example containing a SPARQL query with three triple patterns—namely TP1, TP2 and TP3—and two joins. Consider two different cost-based federation engines with different cardinality estimators. Figure 1a shows the real (Cr) and estimated cardinalities (Ce1 for Engine 1 and Ce2 for Engine

```
SELECT * WHERE {
?s :p1 ?o1 .
Cr(TP1):100
Ce1(TP1):90
Ce2(TP1):200

?s :p2 ?o2 .
Cr(TP2):200
Ce1(TP2):250
Ce2(TP2):600

?s :p3 ?o3 .
Cr(TP3):300
Ce1(TP3):300
Ce2(TP3):500 }
```

(a) Example query



(b) Engine 1 optimal query plan



(c) Engine 2 sub-optimal query plan

Fig. 1.: Motivating Example: A sample SPARQL query and the corresponding query plans of two different federation engines

2) for triple patterns of the query. Let us assume that both engines generate left-deep query plans by selecting triple patterns with the smallest cardinalities to perform their first join. The results of this join are then used to perform the second join with the remaining third triple pattern. By using actual cardinalities, the optimal query execution plan would be to first perform the join between TP1 and TP2 and then perform the second join with TP3. The same plan will be generated by Engine 1 as well, as shown in Figure 1b. The sub-optimal plan generated by Engine 2 is given in Figure 1c. Note that Engine 2 did not select the optimal plan because of large errors in cardinality estimations of triple patterns and joins between triple patterns.

The motivating example clearly shows that good cardinality estimations are essential to produce a better query plan. The question we aim to answer pertains to how much the accuracy of cardinality estimations affects the overall query plan and the overall query runtime performance. To answer this question, the q-error (Q in Figure 1) was introduced in [5] in the database literature. In the next section, we define this measure and propose new metrics based on similarities to measure the overall triple patterns error E_T, overall joins error E_J as well as overall query plan error E_P.

4. Metrics

Now we formally define the q-error and our proposed metrics, namely E_T, E_J, E_P to measure the overall error in cardinality estimations of triple patterns, joins between triple patterns and overall query plan error, respectively.

4.1. q-error

The q-error is the factor by which an estimated cardinality value differs from the actual cardinality value [5].

Definition 1 (q-error). *Let $\vec{r} = (r_1, \ldots, r_n) \in \mathbb{R}$ where $r_i > 0$ be a vector of real values and $\vec{e} = (e_1, \ldots, e_n) \in \mathbb{R}$ be the vector of the corresponding estimated values. By defining $\vec{e}/\vec{r} = \frac{\vec{e}}{\vec{r}} = (e_1/r_1, \ldots, e_n/r_n)$, then q-error of estimation e of r is given as*

$$||e/r||_Q = \max_{1 \leqslant i \leqslant n} ||e_i/r_i||_Q, \text{ where}$$
$$||e_i/r_i||_Q = \max(e_i/r_i, r_i/e_i)$$

In this definition, over- and underestimations are treated symmetrically [5]. In the motivating example given in Figure 1, the real cardinality of TP1 is 100 (i.e., Cr(TP1)=100) while the estimated cardinality by engine 1 for the same triple pattern is 90 (i.e., Cr(TP1) = 90). Thus, the q-error for this individual triple pattern is max(90/100,100/90) = 1.11. The query's overall q-error of its triple patterns (see Figure 1b) is the maximum

value of all the q-error values of triple patterns, i.e., $\max(1.11, 1.25, 1) = 1.25$. The q-error of the complete query plan would be the maximum q-error values in all triple patterns and joins used in the query plan, i.e., $\max(1.11, 1.25, 1, 1.3, 3) = 3$.

The q-error makes use of the ratio instead of an absolute or quadratic difference and is hence able to capture the intuition that only relative differences matter for making planning decisions. In addition, the q-error provides a theoretical upper bound for the plan quality if the q-error of a query is bounded. Since it only considers the maximum value amongst those calculated, it is possible that plans with good average estimations are regarded as poor by this measure. Consider the query plans given in Figure 1b and Figure 1c. Both have a q-error of 3, yet the query plan in Figure 1b is optimal, while the query plan in Figure 1c is not. To solve this problem, we introduce the additional metrics defined below.

4.2. Similarity Errors

The overall similarity error of query triple patterns is defined as follows:

Definition 2 (Triple Patterns Error E_T)**.** *Let Q be a SPARQL query containing triple patterns $T = \{TP_1, \ldots, TP_n\}$. Let $\vec{r} = (Cr(TP_1), \ldots, Cr(TP_n)) \in \mathbb{R}$ be the vector of real cardinalities of T and $\vec{e} = (Ce(TP_1), \ldots, Ce(TP_n)) \in \mathbb{R}$ be the vector of the corresponding estimated cardinalities of T. Then, we define our overall triple pattern error as follows:*

$$E_T = 2 * \frac{|\vec{r}-\vec{e}|}{|\vec{r}|+|\vec{e}|} = 2 * \frac{\sqrt{\sum_{i=1}^{n}(Cr(TP_i)-Ce(TP_i))^2}}{\sqrt{\sum_{i=1}^{n}(Cr(TP_i))^2}+\sqrt{\sum_{i=1}^{n}(Ce(TP_i))^2}}$$

In the motivating example given in Figure 1, the real cardinalities vector $\vec{r}$ = (100,200,300) and the Engine 1 estimated cardinalities vector $\vec{e}$ = (90,250,300). Thus, $E_T = 2 * 0.0658 = 0.1316$. Similarly, Engine 2 estimated cardinality vector is $\vec{e}$ = (200,500,600). Thus, Engine 2 achieves $E_T = 2 * 0.388 = 0.7765$.

Definition 3 (Joins Error E_J)**.** *Let Q be a SPARQL query containing joins $J = \{J_1, \ldots, J_n\}$. Let $\vec{r} = (Cr(J_1), \ldots, Cr(J_n)) \in \mathbb{R}$ a vector of real cardinalities of J and $\vec{e} = (Ce(J_1), \ldots, Ce(J_n)) \in \mathbb{R}$ be the vector of the corresponding estimated cardinalities of J, then the overall joins error is defined as follows:*

$$E_J = 2 * \frac{|\vec{r}-\vec{e}|}{|\vec{r}|+|\vec{e}|} = 2 * \frac{\sqrt{\sum_{i=1}^{n}(Cr(J_i)-Ce(J_i))^2}}{\sqrt{\sum_{i=1}^{n}(Cr(J_i))^2}+\sqrt{\sum_{i=1}^{n}(Ce(J_i))^2}}$$

Definition 4 (Query Plan Error E_P)**.** *Let Q be a SPARQL query and TJ be the set of triple patterns and joins in Q. Let $\vec{r} = (r_1, \ldots, r_n) \in \mathbb{R}$ be a vector of real cardinalities of TJ and $\vec{e} = (e_1, \ldots, e_n) \in \mathbb{R}$ be the vector of corresponding estimated cardinalities of TJ, then the overall query plan error is defined as follows:*

$$E_P = 2 * \frac{|\vec{r}-\vec{e}|}{|\vec{r}|+|\vec{e}|} = 2 * \frac{\sqrt{\sum_{i=1}^{n}(Cr(r_i)-Ce(e_i))^2}}{\sqrt{\sum_{i=1}^{n}(Cr(r_i))^2}+\sqrt{\sum_{i=1}^{n}(Ce(e_i))^2}}$$

In the motivating example given in Figure 1b, the real cardinalities vector of all triple patterns and joins, $\vec{r}$ = (100,200,300,50,50) and the Engine 1 estimated cardinalities vectors $\vec{e}$ = (90,250,300,65,150). Thus, E_P = 2*0.1391 = 0.2784 for Engine 1. Engine 2 achieves E_P = 2*0.3838 = 0.7676. In these matrices, over- and underestimations are also treated symmetrically.

5. Selected Federation Engines

In this section, we give a brief overview of the selected cost-based SPARQL federation engines.

CostFed: CostFed [8] makes use of pre-computed statistics stored in index to estimate the cardinality of triple patterns and joins between triple patterns. CostFed uses both HashJoin and BindJoin for joining the results of triple patterns. Deciding which join to select is based on calculating the cost of both joins. CostFed creates 3 buckets for each distinct predicate used in the RDF dataset. This bucket information is also used during cost calculations and query planning.

SemaGrow: SemaGrow [13] is another cost-based federated query engine. The query planning is based on VoID[2] statistics about datasets. SemaGrow implements bind, hash, and merge joins, and their selection to perform the required join operation is based on a cost calculation. It uses a reactive model for retrieving results of the joins as well as individual triple patterns.

Odyssey: Odyssey [11] makes use of distributed Characteristic sets (CS) [35] and characteristic pair (CP) [36] statistics to estimate cardinalities. It then uses dynamic programming to produce query execution plans.

LHD: LHD [9] is an index-assisted and cardinality-based approach that aims to maximize parallel execution of sub-queries. It also makes use of VoiD statistics to estimate the cardinality of triple patterns and joins

[2] VoID vocabulary: https://www.w3.org/TR/void/

between triple patterns. Bind join is used in the query planning.

SPLENDID: SPLENDID [12] also uses VoID statistics to generate query execution plans. It uses a dynamic programming approach to produce query execution plans.

6. Evaluation and Results

In this section, we discuss our evaluation results. Complete results of our evaluation are also available from our resource homepage. First, we evaluate our novel metrics in terms of how they are correlated with the overall query runtime performances. Thereafter, we compare existing cost-based SPARQL federation engines against the proposed metrics and discuss the evaluation results.

6.1. Experiment Setup and Hardware:

Benchmarks Used: In our experiments, we used the most recent state-of-the-art benchmark for federated engines dubbed LargeRDFBench [26]. The benchmark includes all FedBench[27] queries. LargeRDFBench comprises a total of 40 queries: 14 simple queries (S1-S14) from FedBench, 10 complex queries (C1-C10), 8 complex plus high sources queries (CH1-CH8), and 10 large data queries (L1-L10). We used all queries except large data queries (L1-L10) in our experiments. The reason for skipping L1-L10 was that the evaluation results [26] show that most engines are not yet able to execute these queries. LargeRDFBench comprises a total of 13 real-world RDF datasets of varying sizes. We loaded each dataset into a Virtuoso 7.2 server.

Cost-based Federation Engines: We evaluated five—CostFed [8], Odyessey[11], SemaGrow[13], LHD[9] and SPLENDID[12]—state-of-the-art cost-based SPARQL federation engines. To the best of our knowledge, these are most of the currently available, open-source cost-based federation engines.

Hardware Used: Each Virtuoso was deployed on a physical machine (32 GB RAM, Core i7 processor and 500 GB hard disc). We ran the selected federation engines on a local client machine with same specifications. Our experiments were run in a local environment where the network cost is negligible.

Warm-up and Number of Runs: We warmed up each federation engine for 10 minutes by executing the Linked Data (LD1-LD10) queries from FedBench. Experiments were run 3 times and the results were averaged. The query timeout was set to 30 minutes.

Metrics: We present results for: (1) q-error of triple patterns, (2) q-error of joins between triple patterns, (3) q-error of overall query plans, (4) errors of triple patterns, (5) errors of joins between triple patterns, (6) errors of overall query plans, and (7) the overall query runtimes. In addition, we used Spearman's correlation coefficient to measure the correlation between the proposed metrics and the overall query runtimes. We used simple linear and robust regression models to compute the correlation.

6.2. Regression Experiments

First, we wanted to investigate the dependency between proposed metrics and overall query runtime performance of the federation engines. Figure 2 shows the results of a simple linear regression experiment aiming to compute the dependency between q-error and similarity errors, and the overall query runtimes. For a particular engine, the left figure shows the dependency between the q-error and overall runtime while the right figure in the same row shows the result of the corresponding similarity error. The higher coefficients (dubbed R in the figure) computed in the experiments with similarity errors suggest that it is likely that the similarity errors are a better predictor for runtime. The positive value of the coefficient suggests that an increase in similarity error also means an increase in the overall runtime. It can be observed from the figure that the outliers are contaminating the results. In Figure 3, we further apply robust regression[37–39] using the Huber loss function [40] to remove the outliers from the results (especially for q-errors). This is because we wanted to avoid the possible high impact of outliers. We observe that after removing outliers using robust regression, the similarity-based error correlation further increases. The lower p-values in the similarity-error-based experiments further confirm that our metrics are more likely to be a better predictor for runtime than the q-error. The reason for this result is clear: Our measures exploit more information and are hence less affected by outliers. This is not the case for the q-error, which can be perturbed significantly by a single outlier.

To investigate the correlation between metrics and runtimes further, we measured Spearman's correlation

coefficient between query runtimes and corresponding errors of each of the first six metrics. The results are shown in Table 2.

Table 2 shows that the proposed metrics on average have positive correlations with query runtimes, i.e., the smaller the error, the smaller the query runtimes. The similarity error of overall query plan (E_P) has the highest impact (i.e. 0.35) on query runtimes, followed by the similarity error of the triple pattern (i.e. E_T with 0.27), q-error of joins (i.e. Q_J with 0.26), similarity error of Join (i.e. E_J with 0.22), q-error of overall plan (i.e. Q_P with 0.17), and q-error of triple patterns (i.e. Q_T with 0.06), respectively.

In order to do a fair comparison between the results, we only take the common queries on which every system passed. We eliminate the LHD [9], because it failed in 20/32 benchmark queries, (which is a very high number and only 12 simple queries passed), and is not adequate for comparison. We apply Spearmam's correlation again on common queries. Table 3 shows that the proposed metric has a stronger positive correlation with query runtime when we deal with only common queries. The similarity error of overall plan (E_P) and triple pattern (E_T) has the highest impact (i.e. 0.40) on query runtime, followed by similarity error of joins (i.e. E_J with 0.39), q-error of joins (i.e. Q_J with 0.17) and overall query plan (i.e. Q_P with 0.17), and q-error of triple patterns (i.e. Q_T with 0.01), respectively.

Furthermore, we remove outliers influencing results by applying robust regression. Robust regression is done by Iterated Re-weighted Least Squares (IRLS) [37]. We used Huber weights[40] as weighting function in IRLS. This approach further fine tuned the results and made the correlation for our proposed similarity error and run time stronger. Table 4 shows that all metrics have a stronger positive correlation. However, in our proposed metric this difference is definite. The similarity error of overall query plan (E_P) has the highest impact (i.e. 0.56) on query runtimes, followed by the similarity error of the triple pattern (i.e. E_T with 0.49), similarity error of joins (E_J with 0.45), q-error of joins (i.e. Q_J with 0.22), q-error of overall plan (i.e. Q_P with 0.18) and triple pattern (i.e. Q_P with 0.18), respectively. Table 4 also shows that the q-error for Odyssey is negatively correlated with runtime. We can also observe high q-error values from Figure 4.

Overall, the results show that the proposed similarity errors correlate better with query runtimes than the q-error. Moreover, the correct estimation of the overall plan is clearly the most crucial fragment of the plan generation. Thus, it is important for federation engines to pay particular attention to the cardinality estimation of the overall query plan. However, given that this estimation commonly depends on triple patterns and join estimations, better means for approximating triple patterns and join cardinalities should lead to better plans.

6.3. q-error and Similarity-Based Errors

We now present a comparison of the selected cost-based engines based on the 6 metrics given in Figure 4.

Overall, the similarity errors of query plans given in Figure 4a suggests that CostFed produces the smallest errors followed by SPLENDID, LHD, SemaGrow, and Odyssey, respectively. CostFed produces smaller errors than SPLENDID in 10/17 comparable queries (excluding queries with timeout and runtime errors). SPLENDID produces smaller errors than LHD in 12/14 comparable queries. LHD produces smaller errors than SemaGrow in 6/12 comparable queries. In turn, SemaGrow produces smaller errors than Odyssey in 9/15 comparable queries.

An overall evaluation of the q-error of query plans given in Figure 4b leads to the following result: CostFed produces the smallest errors followed by SPLENDID, SemaGrow, Odyssey, and LHD, respectively. In particular, CostFed produces smaller errors than SPLENDID in 9/17 comparable queries (excluding queries with timeout and runtime error). SPLENDID produces smaller errors than SemaGrow in 9/17 comparable queries. SemaGrow produces smaller errors than Odyssey in 8/13 comparable queries. Odyssey is superior to LHD in 5/8 cases.

An overall evaluation of the similarity error joins leads to a different picture (see Figure 4c). While CostFed remains the best system and produces the smallest errors, it is followed by Odyssey, SPLENDID, SemaGrow, and LHD, respectively. In particular, CostFed outperforms Odyssey in 12/17 comparable queries (excluding queries with timeout and runtime error). Odyssey produces less errors than SPLENDID in 7/14 comparable queries. SPLENDID is superior to SemaGrow in 11/17 comparable queries. SemaGrow outperforms LHD in 7/12 comparable queries.

As an overall evaluation of the q-error of joins given in Figure 4d, CostFed produces the smallest errors followed by SPLENDID, SemaGrow, Odyssey, and LHD, respectively. CostFed produces less errors than SPLENDID in 12/17 comparable queries (excluding queries with timeout and runtime error). SPLENDID produces less errors than SemaGrow in 9/17 comparable queries.

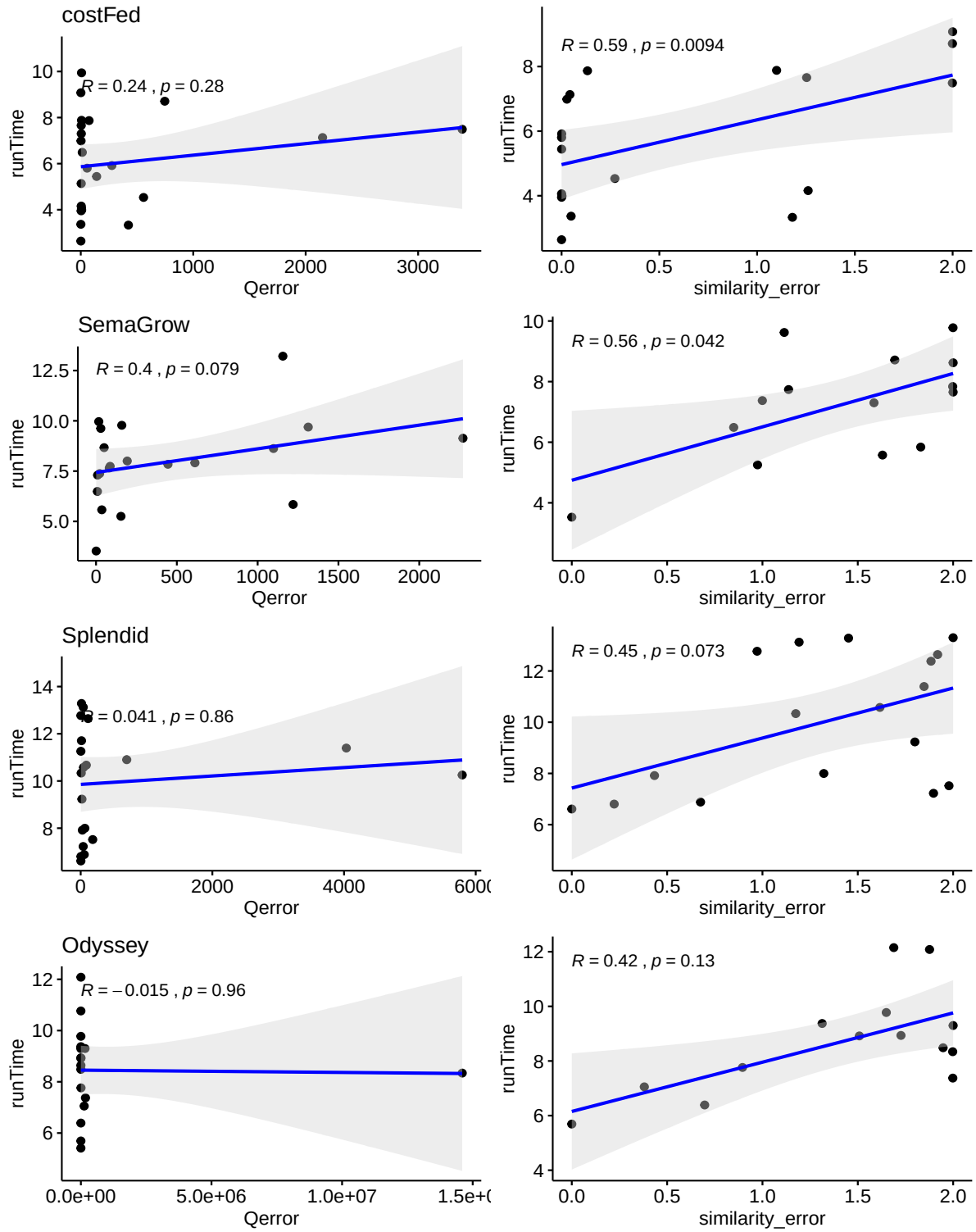


Fig. 2.: q-error and Similarity Error vs. runtime. (Simple Linear Regression Analysis)

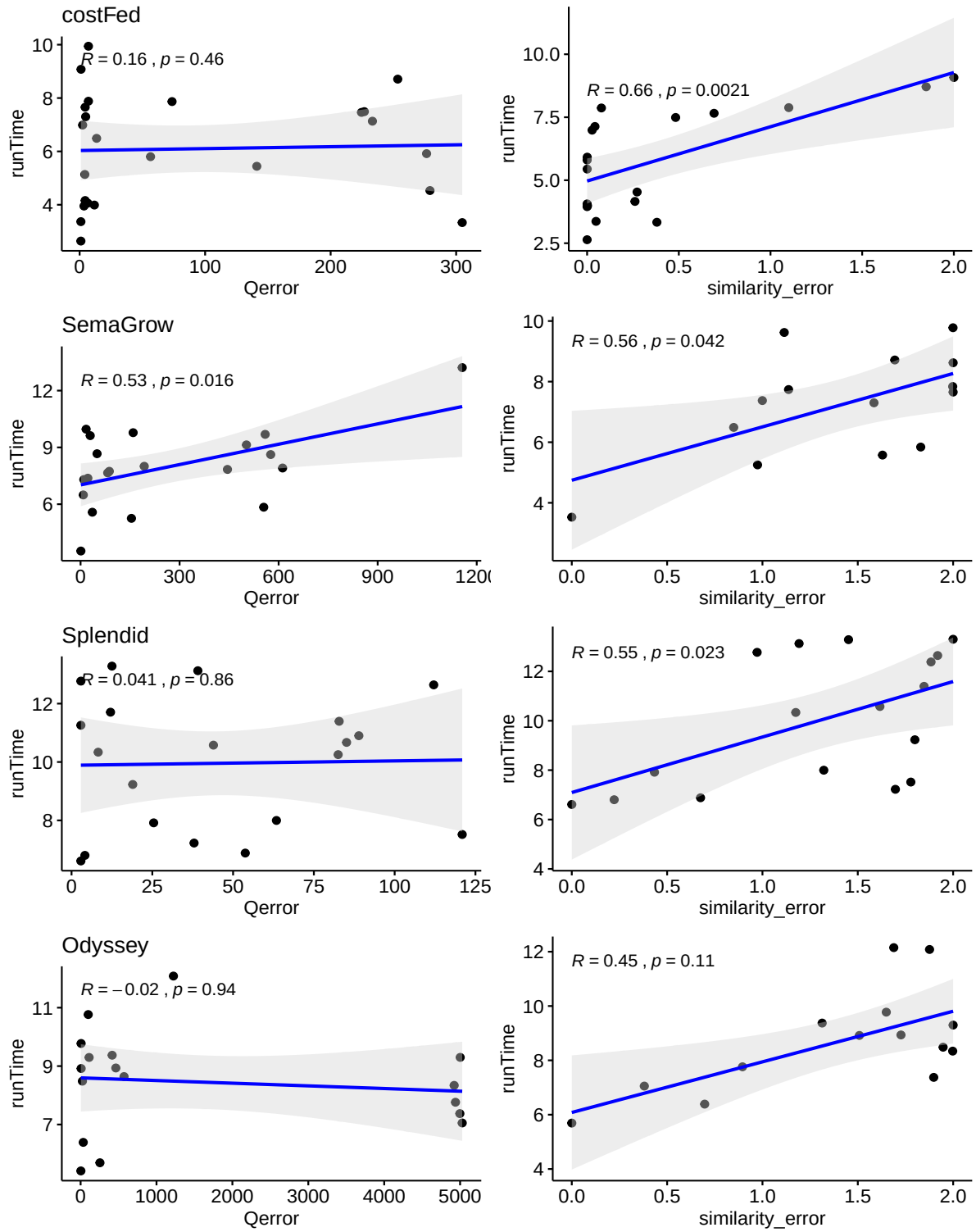


Fig. 3.: q-error and Similarity Error vs. runtime. (Robust Regression Analysis)

	Rank	1	2	3		4	5	6	
		Similarity Error				q-error			
	Feature	E_J	E_P	E_T	Average	Q_J	Q_P	Q_T	Average
F.Q Engines	CostFed	0.23	0.59	0.43	0.42	0.14	0.26	0.1	0.17
	SemaGrow	0.33	0.33	0.33	0.33	0.47	0.37	0.001	0.28
	ODYSSEY	0.11	0.14	0.55	0.26	0.01	0.03	−0.06	−0.01
	SPLENDID	0.3	0.4	0.24	0.32	0.17	0.1	0.24	0.17
	LHD	0.16	0.28	−0.2	0.08	0.51	0.11	0.04	0.22
	Average	0.22	0.35	0.27	0.28	0.26	0.17	0.06	0.17

Table 2: Spearman's rank correlation coefficients between query plan features and query runtimes for all queries.

	Rank	1	2	3		4	5	6	
		Similarity Error				q-error			
	Feature	E_J	E_P	E_T	Average	Q_J	Q_P	Q_T	Average
F.Q Engines	CostFed	0.54	0.61	0.36	0.5	0.11	0.23	0.05	0.13
	SemaGrow	0.44	0.56	0.43	0.48	0.49	0.40	−0.02	0.29
	ODYSSEY	0.22	0.42	0.53	0.39	−0.04	−0.01	−0.20	−0.08
	SPLENDID	0.35	0.45	0.27	0.36	0.12	0.04	0.21	0.12
	Average	0.39	0.51	0.40	0.43	0.17	0.17	0.01	0.12

Table 3: Spearman's rank correlation coefficients between query plan features and query runtimes after linear regression (only for common queries between all systems).

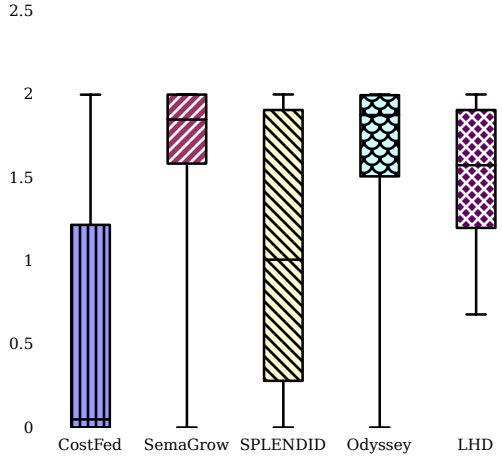
	Rank	1	2	3		4	5	6	
		Similarity Error				q-error			
	Feature	E_J	E_P	E_T	Average	Q_J	Q_P	Q_T	Average
F.Q Engines	CostFed	0.60	0.66	0.62	0.63	0.16	0.16	0.16	0.16
	SemaGrow	0.56	0.56	0.57	0.56	0.60	0.53	0.57	0.56
	ODYSSEY	0.25	0.45	0.59	0.43	−0.04	−0.02	−0.20	−0.08
	SPLENDID	0.49	0.55	0.20	0.38	0.14	0.041	0.18	0.12
	Average	0.45	0.56	0.49	0.50	0.22	0.18	0.18	0.19

Table 4: Spearman's rank correlation coefficients between query plan features and query runtimes after robust regression (only for common queries between all systems). E_J: Similarity Error of Joins, E_P: Similarity Error of overall query plan, E_T: Similarity Error of Triple Patterns, Q_J: q-error of Joins, Q_P: q-error of overall query plan, Q_T: q-error Error of Triple Patterns, F.Q: Federated Query. Correlations and colors (−+): 0.00…0.19 very weak (●●), 0.20…0.39 weak (●●), 0.40…0.59 moderate (●●), 0.60…0.79 strong (●●), 0.80…1.00 very strong (●●).

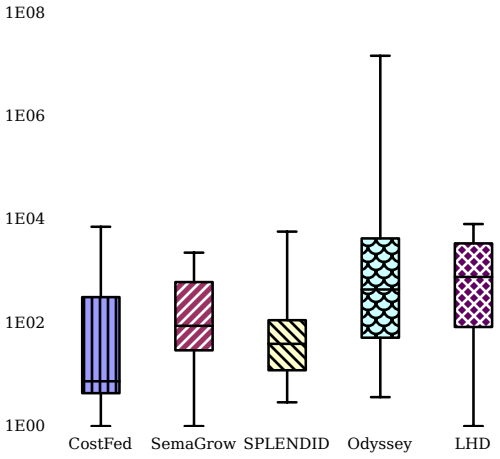
SemaGrow produces less errors than Odyssey in 9/13 comparable queries. Odyssey produces less errors than LHD in 4/8 comparable queries.

Overall, the evaluation of the similarity errors of triple patterns given in Figure 4e reveals that CostFed produces the smallest errors followed by SPLENDID, Odyssey, SemaGrow, and LHD, respectively. CostFed produces smaller errors than SPLENDID in 10/17 comparable queries (excluding queries with timeout and runtime error). SPLENDID produces smaller errors than Odyssey in 15/17 comparable queries. Odyssey produces smaller errors than SemaGrow in 7/14 comparable queries. SemaGrow outperformed LHD in 6/12 queries.

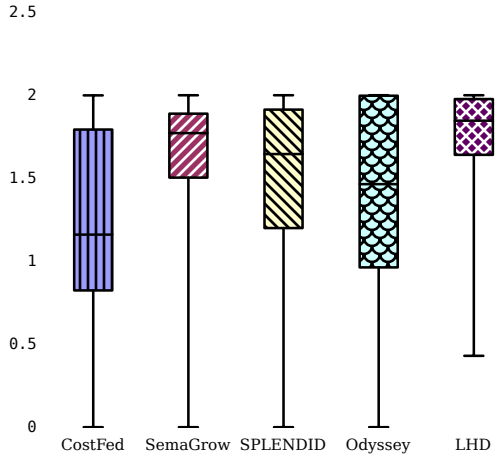
An overall evaluation of the q−error of triple patterns given in Figure 4f leads to a different ranking: CostFed produces the smallest errors followed by LHD, SemaGrow, SPLENDID, and Odyssey, respectively. CostFed outperforms LHD in 6/11 comparable queries (excluding queries with timeout and runtime error). LHD pro-



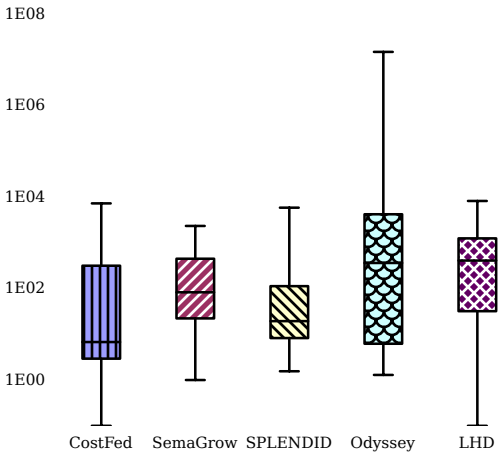
(a) Overall Similarity Error of query plans



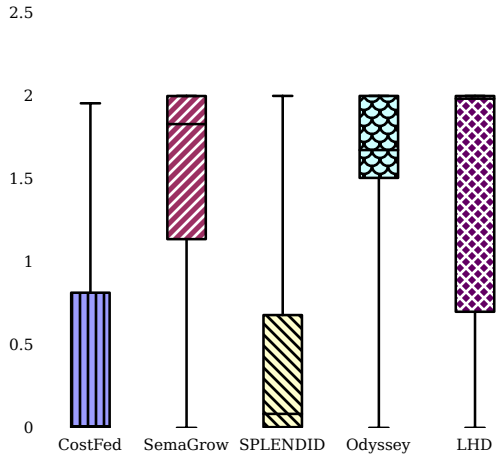
(b) Overall q-error of query plans



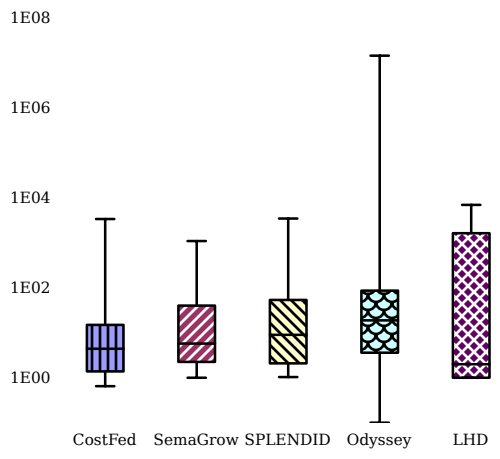
(c) Join Similarity Error of query plans



(d) Join q-error of query plan



(e) Triple pattern Similarity Error of query



(f) Triple pattern q-error of query

Fig. 4.: Similarity and q-error of Query plan

duces fewer errors than SemaGrow in 5/10 comparable queries. SemaGrow is better than SPLENDID in 10/17 comparable queries. SPLENDID produces fewer errors than Odyssey in 7/14 comparable queries.

We observed that it is possible for a federation engine to produce quite a high cardinality estimation error (i.e., 1.974217 is overall similarity error for S11 query in SemaGrow), yet it produces the optimal query plan. Hence, the runtime is smaller (i.e., 191 ms). However, in our evaluation, we compared which engine better estimates the cardinalities, both for joins, as well as triple patterns. Furthermore, these are potentially better metrics for such comparisons.

6.4. Query Execution Time:

Finally, we present the query runtime results of the selected federation engines across the different queries categories of LargeRDFBench. Figure 5 gives an overview of our results. In our runtime evaluation on simple queries (S1-S14) (see Figure 5a), CostFed has the shortest runtimes, followed by SemaGrow, LHD, Odyssey, and SPLENDID, respectively. CostFed's runtimes are shorter than SemaGrow's on 13/13 comparable queries (excluding queries with timeout and runtime error) (average runtime = 524ms for CostFed vs. 2,539ms for SemaGrow). SemaGrow outperforms LHD on 4/11 comparable queries with an average runtime of 2,539ms for SemaGrow vs. 2,752ms for LHD. LHD's runtimes are shorter than Odyssey's on 8/10 comparable queries with an average runtime of 8,515ms for Odyssey. Finally, Odyssey is clearly faster than SPLENDID on 8/12 comparable queries with an average runtime of 131,586.8ms for SPLENDID.

Our runtime evaluation on the complex queries (C1-C10) (see Figure 5b) leads to a different ranking: CostFed produces the shortest runtimes followed by SemaGrow, Odyssey, and SPLENDID, respectively. CostFed outperforms SemaGrow in 6/6 comparable queries (excluding queries with timeout and runtime error) with an average runtime of 3,402ms for CostFed vs. 9,324 for SemaGrow. SemaGrow's runtimes are shorter than Odyssey's in 3/4 comparable queries with an average runtime of 63,157ms for Odyssey. Odyssey is better than SPLENDID in 5/5 comparable queries, where SPLENDID's average runtime is 98,494.52ms.

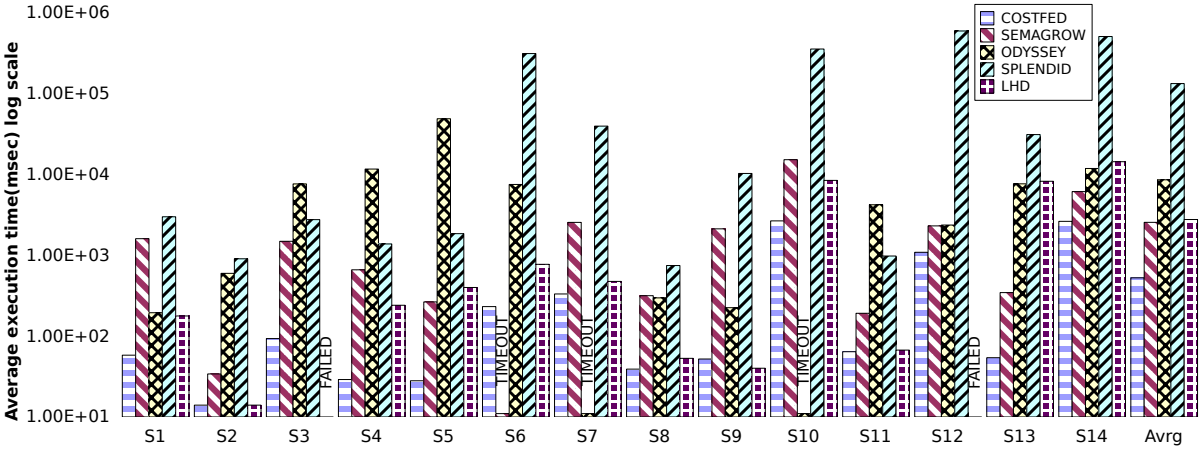
The runtime evaluation on the complex and high sources queries (CH1-C8) given in Figure 5c establishes CostFed as the best query federation engine, followed by SPLENDID and then SemaGrow, respectively. CostFed's runtimes are smaller than SemaGrow in 3/3 comparable queries (excluding queries with timeout and runtime error), with an average runtime of 4,237ms for CostFed vs. 191,315ms for SemaGrow. SPLENDID has no comparable queries with CostFed and SemaGrow. LHD and Odyssey both fail to produce results when faced with complex queries.

7. Conclusion

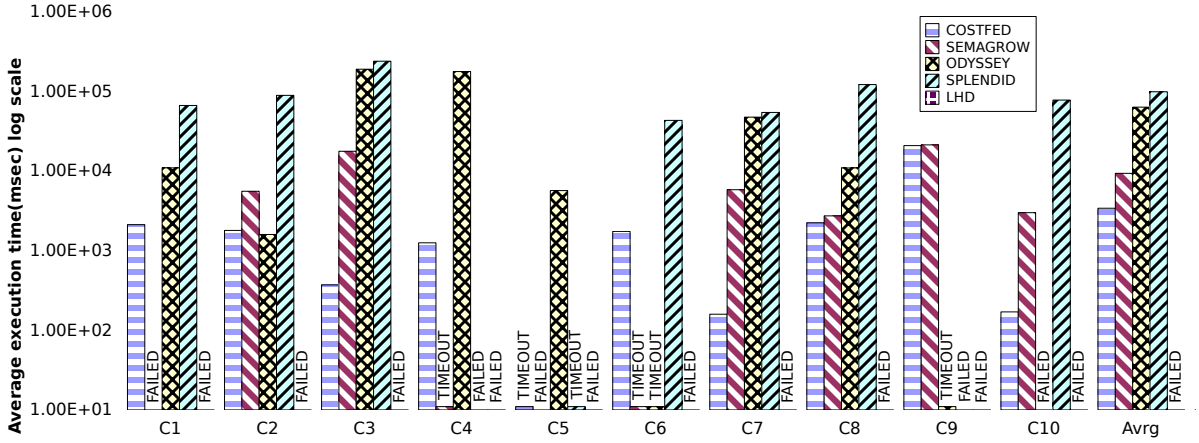
In this paper, we presented an extensive evaluation of existing cost-based federated query engines. We used existing metrics from relational database research and proposed new metrics to measure the quality of cardinality estimators of selected engines. To the best of our knowledge, this work is the first evaluation of cost-based SPARQL federation engines focused on the quality of the cardinality estimations. Overall, our key results are as follows:

- The proposed similarity-based errors have a more positive correlation with runtimes, i.e., the smaller the error values, the better the query runtimes.
- The higher coefficients (R values) with similarity errors, (as opposed to q-error), suggest that the proposed similarity errors are a better predictor for runtime than the q-error.
- The smaller p-values of the similarity errors, as compared to q-error, further confirm that similarity errors are more likely to be a better predictor for runtime than the q-error.
- Errors in the cardinality estimation of triple patterns have a higher correlation to runtimes than the error in the cardinality estimation of joins.Thus, cost-based federation engines must pay particular attention to attaining accurate cardinality estimations of triple patterns
- On average, the CostFed engine produces the fewest estimation errors and has the shortest execution time for the majority of LargeRDFBench queries.

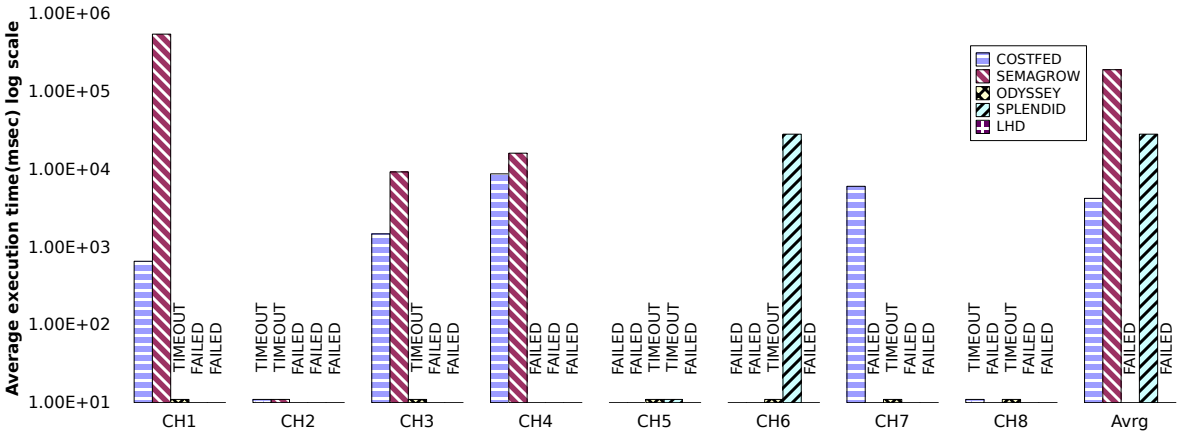
As future work, we want to compare heuristic-based (index-free) federated SPARQL query processing engines with cost-based federated engines. We want to investigate how much an index is assisting a cost-based federated SPARQL engine to generate optimized query execution plans.



(a) Average execution time of simple (S) queries (FedBench)



(b) Average execution time of complex (C) queries (LargeRDFBench)



(c) Average execution time of complex and high data structures (ch) queries(LargeRDFBench)

Fig. 5.: Average execution time of LargeRDFBench and FedBench Queries.

Acknowledgments

This work is supported by the National Research Foundation of Korea(NRF) (grant funded by the Korea government(MSIT) (No. NRF-2018R1A2A2A05023669)). The work has also been supported by the project LIMBO (Grant no. 19F2029I), OPAL (no. 19F2028A), KnowGraphs (no. 860801), and SOLIDE (no. 13N14456) conducted in the University of Leipzig.

References

[1] M. Saleem, Y. Khan, A. Hasnain, I. Ermilov and A.-C. Ngonga Ngomo, A fine-grained evaluation of SPARQL endpoint federation systems, *Semantic Web Journal* **7**(5) (2016), 493–518. doi:10.3233/SW-150186.

[2] N.A. Rakhmawati, J. Umbrich, M. Karnstedt, A. Hasnain and M. Hausenblas, Querying over Federated SPARQL Endpoints - A State of the Art Survey, *CoRR* **abs/1306.1723** (2013). http://arxiv.org/abs/1306.1723.

[3] M. Wylot, M. Hauswirth, P. Cudré-Mauroux and S. Sakr, RDF Data Storage and Query Processing Schemes: A Survey, *ACM Comput. Surv.* **51**(4) (2018), 84:1–84:36. doi:10.1145/3177850.

[4] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper and T. Neumann, How Good Are Query Optimizers, Really?, *Proc. VLDB Endow.* **9**(3) (2015), 204–215. doi:10.14778/2850583.2850594.

[5] G. Moerkotte, T. Neumann and G. Steidl, Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors, *Proc. VLDB Endow.* **2**(1) (2009), 982–993. doi:10.14778/1687627.1687738.

[6] A. Schwarte, P. Haase, K. Hose, R. Schenkel and M. Schmidt, FedX: Optimization Techniques for Federated Query Processing on Linked Data, in: *The Semantic Web – ISWC 2011*, L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy and E. Blomqvist, eds, Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 601–616. ISBN 978-3-642-25073-6. doi:10.1007/978-3-642-25073-6_38.

[7] G. Montoya, M.-E. Vidal and M. Acosta, A Heuristic-based Approach for Planning Federated SPARQL Queries, in: *Proceedings of the Third International Conference on Consuming Linked Data - Volume 905*, COLD'12, CEUR-WS.org, Aachen, Germany, Germany, 2012, pp. 63–74. http://dl.acm.org/citation.cfm?id=2887367.2887373.

[8] M. Saleem, A. Potocki, T. Soru, O. Hartig and A.-C.N. Ngomo, CostFed: Cost-Based Query Optimization for SPARQL Endpoint Federation, Elsevier, 2018, pp. 163–174, Proceedings of the 14th International Conference on Semantic Systems 10th – 13th of September 2018 Vienna, Austria. ISSN 1877-0509. doi:10.1016/j.procs.2018.09.016.

[9] X. Wang, T. Tiropanis and H. Davis, LHD Optimising Linked Data Query Processing Using Parallelisation, in: *Workshop on Linked Data on the Web (LDOWÍ3), Proceedings of the WWW2013*, CEUR Workshop Proceedings, Vol. 996, CEUR-WS.org, Rio de Janeiro, Brazil, 2013. ISSN 1613-0073. http://eprints.soton.ac.uk/350719/.

[10] B. Quilitz and U. Leser, Querying Distributed RDF Data Sources with SPARQL, in: *Proceedings of the 5th European Semantic Web Conference on The Semantic Web: Research and Applications*, ESWC'08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 524–538. ISBN 3-540-68233-3, 978-3-540-68233-2. http://dl.acm.org/citation.cfm?id=1789394.1789443.

[11] G. Montoya, H. Skaf-Molli and K. Hose, The Odyssey Approach for Optimizing Federated SPARQL Queries, *The Semantic Web – ISWC 2017* (2017), 471–489–. ISBN 9783319682884. doi:10.1007/978-3-319-68288-4_28.

[12] O. Görlitz and S. Staab, SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions, in: *Proceedings of the Second International Conference on Consuming Linked Data - Volume 782*, COLD'11, CEUR-WS.org, Aachen, Germany, Germany, 2010, pp. 13–24. http://dl.acm.org/citation.cfm?id=2887352.2887354.

[13] A. Charalambidis, A. Troumpoukis and S. Konstantopoulos, SemaGrow: Optimizing Federated SPARQL Queries, in: *Proceedings of the 11th International Conference on Semantic Systems*, SEMANTICS '15, ACM, New York, NY, USA, 2015, pp. 121–128. ISBN 978-1-4503-3462-4. doi:10.1145/2814864.2814886.

[14] A. Hasnain, R. Fox, S. Decker and H.F. Deus, Cataloguing and Linking Life Sciences LOD Cloud, in: *1st International Workshop on Ontology Engineering in a Data-driven World (OEDW 2012) collocated with 8th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2012)*, 2012, pp. 114–130.

[15] A. Hasnain, S. Sana e Zainab, M. Kamdar, Q. Mehmood, J. Warren ClaudeN., Q. Fatimah, H. Deus, M. Mehdi and S. Decker, A Roadmap for Navigating the Life Sciences Linked Open Data Cloud, in: *Semantic Technology*, T. Supnithi, T. Yamaguchi, J.Z. Pan, V. Wuwongse and M. Buranarach, eds, Lecture Notes in Computer Science, Vol. 8943, Springer International Publishing, 2015, pp. 97–112. ISBN 978-3-319-15614-9. doi:10.1007/978-3-319-15615-6_8.

[16] G. Ladwig and T. Tran, SIHJoin: Querying Remote and Local Linked Data, in: *The Semantic Web: Research and Applications*, G. Antoniou, M. Grobelnik, E. Simperl, B. Parsia, D. Plexousakis, P. De Leenheer and J. Pan, eds, Lecture Notes in Computer Science, Vol. 6643, Springer Berlin Heidelberg, 2011, pp. 139–153. ISBN 978-3-642-21033-4. doi:10.1007/978-3-642-21034-1_10. http://dx.doi.org/10.1007/978-3-642-21034-1_10.

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

[18] I. Abdelaziz, E. Mansour, M. Ouzzani, A. Aboulnaga and P. Kalnis, Lusail: A System for Querying Linked Data at Scale, *Proceedings of the VLDB Endowment* **11**(4) (2017), 485–498. doi:10.1145/3186728.3164144.

[19] K.M. Endris, M. Galkin, I. Lytra, M.N. Mami, M.-E. Vidal and S. Auer, MULDER: Querying the Linked Data Web by Bridging RDF Molecule Templates, in: *Database and Expert Systems Applications (DEXAÍ7)*, D. Benslimane, E. Damiani, W.I. Grosky, A. Hameurlain, A. Sheth and R.R. Wagner, eds,

Springer International Publishing, Cham, 2017, pp. 3–18. ISBN 978-3-319-64468-4. doi:10.1007/978-3-319-64468-4_1.

[20] M. Saleem and A.-C. Ngonga Ngomo, HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation, in: *The Semantic Web: Trends and Challenges*, V. Presutti, C. d'Amato, F. Gandon, M. d'Aquin, S. Staab and A. Tordai, eds, Lecture Notes in Computer Science, Vol. 8465, Springer International Publishing, 2014, pp. 176–191. ISBN 978-3-319-07442-9. doi:10.1007/978-3-319-07443-6_13.

[21] M. Saleem, A.-C. Ngonga Ngomo, J. Xavier Parreira, H.F. Deus and M. Hauswirth, DAW: Duplicate-AWare Federated Query Processing over the Web of Data, in: *Proceedings of the 12th International Semantic Web Conference - Part I*, Lecture Notes in Computer Science, Springer-Verlag New York, Inc., New York, NY, USA, 2013, pp. 574–590. ISBN 978-3-642-41334-6. doi:10.1007/978-3-642-41335-3_36.

[22] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo and E. Ruckhaus, ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints, in: *The Semantic Web – ISWC 2011*, L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy and E. Blomqvist, eds, Lecture Notes in Computer Science, Vol. 7031, Springer-Verlag Berlin Heidelberg, 2011, pp. 18–34. ISBN 978-3-642-25072-9. doi:10.1007/978-3-642-25073-6_2.

[23] J. Umbrich, A. Hogan, A. Polleres and S. Decker, Link Traversal Querying for a Diverse Web of Data, *Semantic Web Journal* **6**(6) (2015), 585–624. doi:10.3233/SW-140164.

[24] M. Acosta, M.-E. Vidal and Y. Sure-Vetter, Diefficiency Metrics: Measuring the Continuous Efficiency of Query Processing Approaches, in: *The Semantic Web – ISWC 2017*, C. d'Amato, M. Fernandez, V. Tamma, F. Lecue, P. Cudré-Mauroux, J. Sequeda, C. Lange and J. Heflin, eds, Springer-Verlag Berlin Heidelberg, Cham, 2017, pp. 3–19. ISBN 978-3-319-68204-4.

[25] A. Hasnain, Q. Mehmood, S. Sana E Zainab, M. Saleem, C. Warren Jr, D. Zehra, S. Decker and D. Rebholz-Schuhman, BioFed: Federated query processing over life sciences linked open data, *Journal of Biomedical Semantics* **8**(1) (2017), 13. doi:10.1186/s13326-017-0118-0.

[26] M. Saleem, A. Hasnain and A.-C. Ngonga Ngomo, LargeRDFBench: A Billion Triples Benchmark for SPARQL Endpoint Federation, *Journal of Web Semantics* **48** (2018), 85–125. doi:10.1016/j.websem.2017.12.005.

[27] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte and T. Tran, FedBench: A Benchmark Suite for Federated Semantic Data Query Processing, in: *The Semantic Web – ISWC 2011*, L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy and E. Blomqvist, eds, Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 585–600. ISBN 978-3-642-25073-6. doi:10.1007/978-3-642-25073-6_37.

[28] O. Görlitz, M. Thimm and S. Staab, SPLODGE: Systematic Generation of SPARQL Benchmark Queries for Linked Open Data, in: *Proceedings of the 11th International Conference on The Semantic Web - Volume Part I*, The Semantic Web – ISWC'12, Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 116–132. ISBN 978-3-642-35175-4. doi:10.1007/978-3-642-35176-1_8.

[29] M. Morsey, J. Lehmann, S. Auer and A.-C.N. Ngomo, DBpedia SPARQL Benchmark: Performance Assessment with Real Queries on Real Data, in: *Proceedings of the 10th International Conference on The Semantic Web - Volume Part I*, The Semantic Web – ISWC'11, Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 454–469. ISBN 978-3-642-25072-9. doi:10.1007/978-3-642-25073-6_29.

[30] N.A. Rakhmawati, M. Saleem, S. Lalithsena and S. Decker, QFed: Query Set For Federated SPARQL Query Benchmark, in: *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '14, ACM, New York, NY, USA, 2014, pp. 207–211. ISBN 978-1-4503-3001-5. doi:10.1145/2684200.2684321.

[31] A. Hasnain, M. Saleem, A.N. Ngomo and D. Rebholz-Schuhmann, Extending LargeRDFBench for Multi-Source Data at Scale for SPARQL Endpoint Federation, in: *Proceedings of the 12th International Workshop on Scalable Semantic Web Knowledge Base Systems co-located with 17th International Semantic Web Conference, SSWS@ISWC 2018, Monterey, California, USA, October 9, 2018*, Vol. 2179, 2018, pp. 28–44. http://ceur-ws.org/Vol-2179/SSWS2018_paper3.pdf.

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

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

[34] M. Schmidt, T. Hornung, G. Lausen and C. Pinkel, SPˆ 2Bench: A SPARQL Performance Benchmark, in: *Proceedings of the 25th International Conference on Data Engineering ICDE*, IEEE, 2009, pp. 222–233.

[35] T. Neumann and G. Moerkotte, Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins, in: *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, IEEE, 2011, pp. 984–994, IEEE Computer Society. ISSN 1063-6382. doi:10.1109/ICDE.2011.5767868.

[36] A. Gubichev and T. Neumann, Exploiting the query structure for efficient join ordering in SPARQL queries., in: *EDBT*, Vol. 14, 2014, pp. 439–450.

[37] P.W. Holland and R.E. Welsch, Robust regression using iteratively reweighted least-squares, *Communications in Statistics - Theory and Methods* **6**(9) (1977), 813–827. doi:10.1080/03610927708827533.

[38] D.P. O'Leary, Robust Regression Computation Using Iteratively Reweighted Least Squares, *SIAM J. Matrix Anal. Appl.* **11**(3) (1990), 466–480. doi:10.1137/0611032.

[39] P.J. Rousseeuw and A.M. Leroy, *Robust regression and outlier detection*, Vol. 589, 1st edn, John wiley & sons, Inc., New York, NY, USA, 1987. ISBN 0-471-85233-3. doi:10.1002/0471725382.

[40] P.J. Huber, *Robust Estimation of a Location Parameter*, in: *Breakthroughs in Statistics: Methodology and Distribution*, S. Kotz and N.L. Johnson, eds, Springer New York, New York, NY, 1992, pp. 492–518. ISBN 978-1-4612-4380-9. doi:10.1007/978-1-4612-4380-9_35.