


Challenges of source selection in the WoD

Tobias Grubenmann  (orcid.org/0000-0003-0391-584X),
Abraham Bernstein (orcid.org/0000-0002-0128-4602),
Dmitry Moor (orcid.org/0000-0002-4658-3489), and
Sven Seuken (orcid.org/0000-0001-8525-8120)

Department of Informatics, University of Zurich, Switzerland,
{grubenmann, bernstein, dmoor, seuken}@ifi.uzh.ch

Abstract. Federated querying, the idea to execute queries over several distributed knowledge bases, lies at the core of the semantic web vision. To accommodate this vision, SPARQL provides the SERVICE keyword that allows one to allocate sub-queries to servers. In many cases, however, data may be available from multiple sources resulting in a combinatorially growing number of alternative allocations of subqueries to sources. Running a federated query on all possible sources might not be very lucrative from a user’s point of view if extensive execution times or fees are involved in accessing the sources’ data. To address this shortcoming, federated join-cardinality approximation techniques have been proposed to narrow down the number of possible allocations to a few most promising (or results-yielding) ones.

In this paper, we analyze the usefulness of cardinality approximation for source selection. We compare both the runtime and accuracy of Bloom Filters empirically and elaborate on their suitability and limitations for different kind of queries. As we show, the performance of cardinality approximations of federated SPARQL queries degenerates when applied to queries with multiple joins of low selectivity. We generalize our results analytically to any estimation technique exhibiting false positives. These findings argue for a renewed effort to find novel join-cardinality approximation techniques or a change of paradigm in query execution to settings, where such estimations play a less important role.

Keywords: Approximate Query Processing · Bloom Filter · Federated SPARQL · Source Selection · Web of Data

1 Introduction

At the core of the Semantic Web vision lies the possibility to ubiquitously access distributed, machine-readable, linked data. This Web of Data (WoD) relies on the notion of being able to access partial information from a variety of sources that then gets combined to an integrated answer.

One major approach to achieving this functionality in a distributed fashion is federated querying [2, 3, 7, 14, 18, 24, 25]. It relies on traditional database approaches to join partial results from multiple sources into a combined answer.

Specifically, it divides a query into subqueries and delegates the execution of each of these subqueries to one or more remote databases, which on the WoD are called endpoints. A query execution plan assigns the subqueries to a certain set of endpoints and determines the order of the subquery execution. Hereby, results from one subquery can vastly reduce the computational effort of answering another. One major problem of querying the Web of Data is *source selection*, which is deciding which subqueries should be delegated to which SPARQL endpoints during query execution and which endpoints should not be considered for query execution at all. We will focus in this paper on the cardinality of the query answer as the metric to evaluate the worthiness of including certain sources into query execution.

Ideally, a user would be able to estimate the cardinality of the query answer for any subset of all relevant sources. Given the knowledge about the resulting cardinality for different combinations of sources, a user could make an informed decision whether a certain source should be included into the federated query execution or not. By an informed decision we mean deciding whether selecting and accessing a certain subset of all available endpoints is worth the time and, potentially, fees which are associated with accessing these endpoints.

In this paper, we argue that **the performance of *cardinality approximations of federated SPARQL queries degenerates when applied to queries with multiple joins having low join selectivities***. This means that such approximations are not sufficiently precise to allow a user to make an informed decision. As a consequence, a user who cannot afford to query all relevant sources for a given query must blindly exclude some relevant sources risking low cardinality or empty query answers, even though solutions to the query would be available on the WoD. Specifically, our contributions are:

- We show empirically that the *cumulative error of cardinality estimation techniques based on Bloom Filters explodes* in the combinatorial distributed setting of the WoD, which questions its usefulness for informed source selection.
- We show empirically that the explosion of the *cumulative error often makes join-cardinality estimation slower than executing the actual query*. Hence, using such a technique may not only lead to suboptimal results but even slow down the query execution process, which is exactly the opposite of the goal of source selection.
- Using a theoretical analysis of the problem, we explain why these *negative results necessarily occur when using any estimation technique exhibiting false positives* in combination with queries having low join selectivities.

The remainder of this paper is organized as follows. First, we succinctly discuss the most relevant related work. Next, Section 3 provides empirical evidence of our claims about the limited usefulness of join approximation techniques using Bloom Filters, which is followed by a discussion of the results. In Section 4, we present our main result: a theoretical analysis which explains the cumulative error and associated runtime behavior that federated cardinality approximation techniques face in the WoD. We close with some conclusions.

2 Related Work

Federated SPARQL querying and source selection: Different approaches have been proposed to query RDF data in a federated setting. Mediator systems like FedX [24] and DARQ [20] allow a user to query a federation of endpoints in a transparent way while incorporating all known SPARQL endpoints into the query answer. The federation appears to the user as one big SPARQL endpoint holding the data of all the members of the federation. Once the members are specified and initialized, the user can issue SPARQL queries against the mediator without having to adapt the query for federated execution or providing any additional information about the federation members.

Avalanche [3] and ANAPSID [2] propose different, more dynamic systems where they relax the requirement of complete results and allow certain endpoints to fail. Their systems focus on robustness of query execution in the Web. Avalanche [3] executes all possible queries (i.e., all combinations of possible endpoints) in parallel eventually timing out a query when the rate of incoming results slows down. In queries with many combinations this may lead to a very high network load and a significant time between querying and query completion. ANAPSID [2], in contrast, runs only one query plan and dispatches each sub-query to every possible endpoint using a mediator. This results in a highly robust execution but again, faces the danger of including a very large number of endpoints if no sensible source-selection approach is available.

SPLENDID [9] proposed to exploit service descriptions and VOID statistics about each endpoint to perform source selection and query optimization. HiBISCuS [21] uses join-aware techniques to select relevant sources for federated query execution. HiBISCuS maintains an index which stores the authorities of certain URIs. [27] introduced Fed-DSATUR, an algorithm for SPARQL query decomposition in federated settings. They do not use statistics, indices, or estimates for source selection.

The SPARQL 1.1 Federated Query extension [10] follows a different approach: a user must explicitly specify which part of the query should be executed on which server. The extension requires the user to know which SPARQL endpoint can provide data for which subquery and rewrite the query accordingly using a special SERVICE-clause.

Duplicate aware source selection [22] tries to eliminate sources with duplicate data using Min-Wise Independent Permutations. [11] used Bloom Filters for source selection of RDF sources and investigated the number of requests needed for an approximation to achieve a certain recall.

Good estimates of the contribution of different sources towards a query answer plays an important role in [16] and [17], where users have to pay for accessing the selected sources.

In contrast to the work presented so far, we perform an empirical and theoretical analysis of the error behavior for the problem of source selection when the cardinality of the result is used as the deciding factor.

Cardinality Estimation Techniques:

In the traditional database domain, join approximation has been used as a suitable technique for approximate query processing [8]. The goal of approximate query processing is to compute an answer that approximates the query answer without having to execute the query. Join approximations can be used to calculate the expected cardinality and the join selectivity of a specific query.

A variety of approaches provide *data synopses* (i.e., summaries of the data) for join approximation. Histograms [13] and Wavelets [8] have been used to approximate the distribution of a dataset over a given domain. Also, Bloom Filters were first proposed as a space-efficient probabilistic data structure to approximate sets [5]. The advantage of Bloom Filters is that they allow to specify the desired false-positive rate for set-membership checking without leading to false-negatives. Given that they also allow intersections between bloom-filtered sets they have become a de-facto standard for join approximations. Q-Trees [19] were introduced as a special data summary technique for RDF data. [26] compared the runtime and space complexity of indexing techniques, multidimensional histograms, and Q-Trees and evaluated, in particular, their usefulness for source selection and highlighted the superiority of Q-Trees over the others. Sampling methods [15] do not rely on a synopsis but on a selection of the data. Hence, they do not produce false positive matches but might produce false negatives. Sampling methods provide a lower bound on the cardinality of a join.

Join synopses [1] are special summary structures built for join approximation. They are constructed for specific, ex-ante known join operations and are therefore not suitable to the purely ad-hoc federated settings. They are, however, useful when one knows that certain joins are likely to occur.

Finally, [12] studied the propagation of errors in the size of the join result. In this paper, we will extend the analysis done by [12] to the domain of SPARQL queries.

3 Experimental Evaluation of the Cumulative Join Estimation Error

The goal of this section is to show the relative error and runtime behavior of join cardinality approximation using Bloom Filters, which motivated our theoretical analysis of the problem and our conclusion that join approximation techniques are problematic for source selection. We used Bloom Filters for the approximation as they provide an easy and straightforward way to encode strings like IRIs and Literals.

In the following, we will first describe the experimental setup, including the query approximation engine and the data we used before presenting the results.

3.1 Query Approximation Engine

We implemented a query engine that allows us to execute joins over federated SPARQL endpoints on dynamically generated data synopses. The query engine accepts a query consisting of basic graph patterns using the SPARQL 1.1

SERVICE-clause to allocate a certain Basic Graph Pattern (BGP), called *service pattern*, to specified endpoints. Our approximation engine currently does not yet support UNION-clauses, OPTIONAL-clauses and filters outside of service patterns.

To approximate a join between two service patterns, a data synopsis of the data matching the first service pattern is generated by the responsible endpoint. This synopsis summarizes the bindings of the joining variables for each solution of the assigned service pattern. The data synopsis is generated by inserting the string representation of the bindings of a solution into a Bloom Filter. If multiple variables are joining, the bindings are combined into one string using a special delimiter. The endpoint responsible for the second SERVICE-clause receives the data synopsis and does a membership check on the string representation of the bindings of the joining variables of its assigned service pattern. The bindings for which the membership check is positive form the basis for the *join synopsis*. The join synopsis summarizes the bindings of those variables which are joining with the next service pattern and is used as input for the next join approximation step.

To illustrate the approximation process, Figure 1 shows how the query in Listing 1 would be approximated. First, `ep1.com` receives the first service pattern, consisting of only one triple pattern `?a ex:p ?x`, and creates a list of bindings for variable `?a` (① in Figure). These bindings get approximated by an appropriate data synopsis (②). The synopsis is joined with the bindings provided by endpoint `ep2.com` for the second service pattern `?a ex:p ?b` (③). Note that only variable `?a` is involved in the join while a synopsis for the corresponding bindings for variable `?b` is created (④). The second synopsis is joined with the bindings for the third service pattern `?y ex:p ?b` (⑤). Since this clause is the last one, there is no further synopsis needed. Instead, we count the number of bindings that join with this last synopsis (⑥). This number is the estimated cardinality of the join between the three service patterns when they are assigned to the sources according to the federated query in Listing 1.

Listing 1. A SPARQL query with 3 Service Patterns, each consisting of 1 Triple Pattern.

```
PREFIX ex: <http://example.com/>
SELECT * WHERE {
  SERVICE <http://ep1.com> {
    ?a ex:p ?x . }
  SERVICE <http://ep2.com> {
    ?a ex:q ?b . }
  SERVICE <http://ep3.com> {
    ?y ex:r ?b . }
}
```

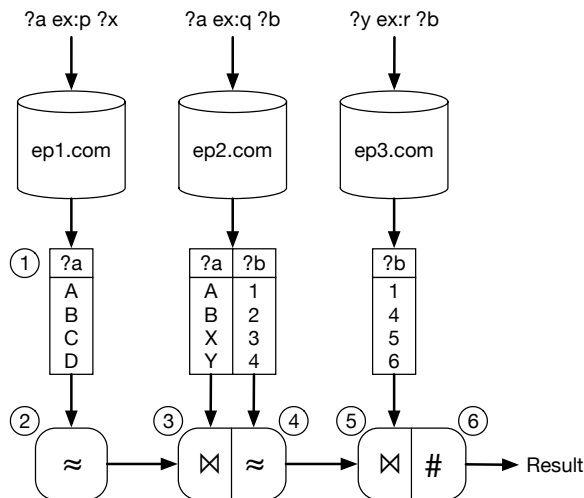


Fig. 1. Approximating the query in Listing 1 using our approximation engine.

3.2 Experimental Setup

For our evaluation, we investigated the scenario where each triple pattern must be sent to a different source. This means that it is not possible to form exclusive groups to speed up query processing/approximation, as proposed by [24].

For the evaluation, we used FedBench [23] as a benchmark. FedBench consists of 25 queries and more than 200 million triples distributed over 9 datasets. Since we do not support UNION or OPTIONAL-clauses at the moment, we removed queries containing those clauses from our evaluations. To give a baseline for the execution time of the different approximation techniques, we executed each query using the query engine Jena ARQ¹. We used the SERVICE-clause to direct each triple pattern to a separate SPARQL endpoint. We used Blazegraph² as a triple store. Table 1 shows the queries used, their runtime in milliseconds when using Jena ARQ, the actual cardinality of the query answer, and the number of triple patterns in the query.

We simulated both, query execution using Jena ARQ and the approximations using Bloom Filters, with a network speed of 10 Mbps, which is around the average speed of the top 10 countries in the world [4]. For the query execution, we adapted Jena ARQ to do block-nested-loop joins with a block size of 500 bindings to reduce the number of HTTP-connections, which have a negative impact on the runtime behavior of federated query execution. In addition, we optimized the join order of the different queries using simple heuristics to keep

¹ <https://jena.apache.org>

² <https://www.blazegraph.com>

query execution in a reasonable time-frame. The query execution and all query approximations used the same join ordering to keep the results comparable.

For the Bloom Filter implementation, we used the Guava Google Core Library for Java³.

Table 1. The execution time, count, and number of triple patterns of the different queries.

Query	Time [ms]	Cardinality	Triple patterns
CD3	2.50E+03	2	5
CD4	3.90E+02	1	5
CD5	4.80E+02	2	4
CD6	1.30E+03	11	4
CD7	5.70E+02	1	4
LS3	3.80E+04	9054	5
LS4	5.20E+02	3	7
LS5	1.02E+02	393	6
LS6	4.40E+05	28	5
LD1	6.90E+02	308	3
LD2	4.70E+02	185	3
LD3	7.60E+02	159	4
LD4	3.10E+03	50	5
LD5	3.00E+02	28	3
LD6	6.20E+02	39	5
LD7	1.50E+03	1216	2
LD8	5.90E+02	22	5
LD9	3.20E+02	1	3
LD10	2.90E+02	3	3
LD11	2.00E+03	376	5

3.3 Results

Figure 2 shows the absolute value of the *relative* error and the *relative* execution time of the approximation both computed with respect to the actual runtime and count when running the query in a federated fashion using Jena ARQ.

The relative error e_{rel} is defined as

$$e_{rel} = \frac{card_{est} - card_{actual}}{card_{actual}},$$

where $card_{est}$ is the estimated cardinality of the query answer based on the approximation and $card_{actual}$ is the actual cardinality of the query answer.

³ <https://github.com/google/guava>

The relative execution time t_{rel} is defined as

$$t_{rel} = \frac{t_{est}}{t_{actual}},$$

where t_{est} is the runtime of the approximation technique and t_{actual} is the runtime of the query execution using Jena ARQ.

Clearly, a relative runtime of less than 1 is desirable, as otherwise it would be faster to execute the query and get the actual cardinality. For the relative error, it is not so clear what kind of error would still be in an acceptable range.

Each plot in Figure 2 shows the relative error (solid line) and relative execution time (dashed line). We measured the error and execution time for false positive rates of $fpp = 0.1, 0.01, 10^{-4}, 10^{-8}$.

As we can see in Figure 2, the runtime of the Bloom Filter approximation is very often disappointing. The approximation tends to require considerably more time for the approximation than the actual query execution. Surprisingly, the execution time for those approximations often improves when increasing the size of the underlying data synopsis. The discussions in Section 4 provide a good explanation for this behavior: the more accurate the synopsis, the less false positives must be processed. The overhead in processing more false positives seem to have a bigger negative impact on the runtime than the reduction of the size of the synopsis. *This behavior somewhat counteracts the actual purpose of a data synopsis to provide a trade-off between less accurate information and reduced processing time.*

Discussion of selected queries: The Bloom Filter approximation shows good results for the runtime of queries LS3, LS5, LS6, LD2, and LD4. Also, the error is comparably low and most of the time below 1. For those queries, the approximation can be considered successful: the approximation is able to return a reasonable approximation of the result size while running considerably faster than the actual query execution.

Queries CD7, LS4, and LD11 show worse approximation for a false positive probability of 10^{-8} than for a probability of 10^{-4} . One likely explanation for this is the fact that the original false positive analysis done by [5] is incomplete and only gives a lower bound on the false positive rate. Indeed, as [6] points out, the actual false positive rate might be worse than expected when a small value for the false-positive probability is chosen as a parameter and a large number of hash functions have to be used in the filters.

The approximation yields a relative error of 0 for the query LD9. The reason for this behavior is that the last triple pattern only matches one single triple. Thus, our approximation engine predicts a cardinality of at most 1, because the prediction is based on the number of those triples matching the last triple pattern which also join the synopsis of the previous joins, which can never be larger than the number of triples matching the last triple pattern. At the same time, the actual result of the query is also 1. Hence, approximation technique which overestimate the cardinality will yield a perfect prediction, necessarily. However, the relative runtime of the approximation methods is around 1.

The query LD4 is another one where the last triple pattern only matches one single triple. Again, our approximation engine predicts a cardinality of at most 1. But this time, the actual result is not 1 but 50. In fact, all 50 different results have the same binding for the last joining variable. As the Bloom Filter does not account for duplicated values the approximation wrongly predicts 1 instead of 50. At the same time, the approximation speed profits slightly from this error by yielding a faster execution time.

4 Theoretical Analysis of the Cumulative Join Estimation Error

In this section, we investigate the theoretical foundations which can explain the disappointing performance of our Bloom Filter join approximation. We will estimate the cumulative error for WoD queries for approximation techniques that overestimate the results due to false positives, which includes all data synopses which are not based on sampling, in particular, our Bloom Filter-based method. Such overestimating data synopses can lead to false-positive matches (i.e., the prediction of a match where there is none) due to loss of information. When approximating multiple joins, the result of the first join (including its false positives) is again encoded as a data synopsis passed to the second join, which will now attempt to match all encoded elements including the false-positives. Hence, the error of the synopsis gets propagated through each join and accumulates [12].

We now formally discuss the propagation of the error in a multi-join, that is, a sequence of joins where the output of one join is an input for the next join. For this we extend the formula for the error derived by [12] by analyzing the relation between the rate of false positive matches and the join selectivity based on the following assumption:

Assumption 1 *All joins are equality inner-joins.*

Assumption 1 is motivated by the fact that we do not consider filter expressions in our evaluation and hence, we only support equality joins. We will not discuss outer joins because their cardinality estimation is trivial.

Assume we want to approximate the join result of joining $m + 1$ basic graph patterns bgp_0, \dots, bgp_m . We define n_i for $i \in \{0, \dots, m\}$ as the number of results selected by BGP bgp_i from the corresponding dataset. Let n_i^{FP} be the number of false positives at step i , which is the number of elements that are wrongly classified as a match given the synopsis from the previous joins. We define the false positive rate fpr_i as the ratio between n_i^{FP} and n_i .

Let $prop_i^{FP}$ for $i \in \{1, \dots, m\}$ be the *propagation rate* of the false positives in the synopsis for the join approximation between bgp_0, \dots, bgp_{i-1} . The propagation rate indicates how many false-positives matches are produced *on average* by a single false-positive propagated from previous join approximations.

The expected number of false positives FP_k for the approximation of the join of bgp_0, \dots, bgp_k is the number of false positives introduced by fpr_k for bgp_k plus

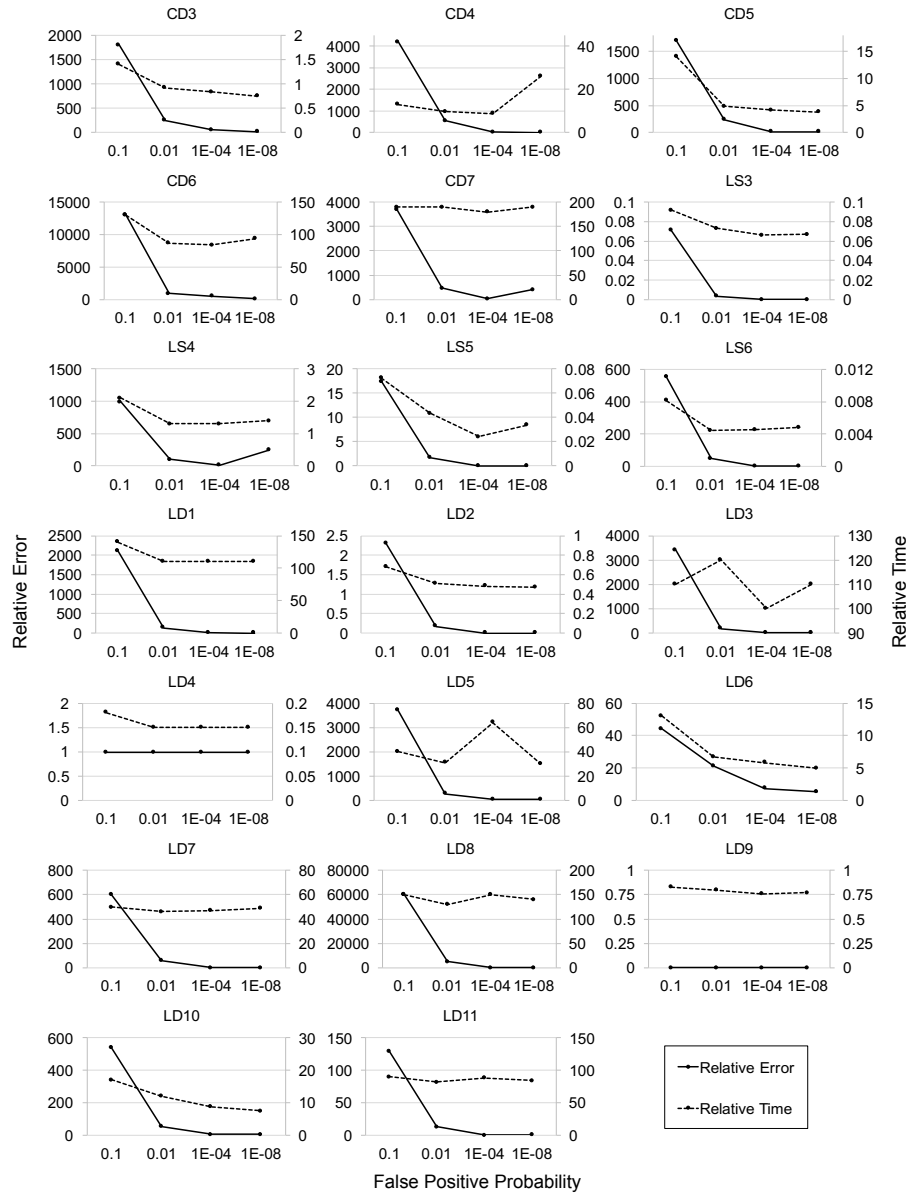


Fig. 2. Relative error (left vertical axis) and relative execution time (right vertical axis) for different false positive probabilities.

the false-positives given the false-positives FP_{k-1} of the approximation of the join of bgp_0, \dots, bgp_{k-1} :

$$FP_k = \underbrace{fpr_k \cdot n_k}_{\text{synopsis error}} + \underbrace{prop_k^{FP} \cdot FP_{k-1}}_{\text{propagated error}} . \quad (1)$$

We define $FP_0 := 0$, as there is no propagated error influencing the first join operation. Applying Equation 1 recursively gives the following formula for the number of false positives FP_m of the approximation of the join of bgp_0, \dots, bgp_m :

$$FP_m = \sum_{i=1}^m fpr_i \cdot n_i \cdot \prod_{j=i+1}^m prop_j^{FP} . \quad (2)$$

To compare the number of false positive matches with the number of true positive matches, we analogously compute the number of true positives TP_m . To do that we define the propagation rate of the true-positives $prop_i^{TP}$ for the join between bgp_0, \dots, bgp_{i-1} (just like we defined the propagation rate $prop_i^{FP}$ for false-positives). Using this definition, the number of true positives TP_k of joining bgp_0, \dots, bgp_k is:

$$TP_k = n_0 \cdot \prod_{j=1}^k prop_j^{TP} . \quad (3)$$

To continue our analysis, we introduce the following assumption:

Assumption 2 *False-positive matches and true-positive matches have the same propagation rate, i.e. $prop_j^{FP} = prop_j^{TP} =: prop_j$.*

Assumption 2 is motivated by the fact that the propagation rate of both, true positives and false positives, are influenced by the type of URI and not by the fact whether they are false or true positive. For example, there is an average number of addresses joining with a person, independent of whether the person is true or false positive. Hence, we assume that there is no bias which would cause that a true positive match has, on average, a lower/higher propagation rate than a false positive match.

Under Assumption 2, we get the following formula for the relative error E of the approximation:

$$\begin{aligned} E = \frac{FP_m}{TP_m} &= \frac{\sum_{i=1}^m fpr_i \cdot n_i \cdot \prod_{j=i+1}^m prop_j}{n_0 \cdot \prod_{j=1}^m prop_j} \quad (4) \\ &= \sum_{i=1}^m \frac{fpr_i \cdot n_i}{n_0 \cdot \prod_{j=1}^i prop_j} . \end{aligned}$$

We define the selectivity $sel_{bgp_0, \dots, bgp_j}$ of the join between bgp_0, \dots, bgp_j as the number of results of the join divided by the product $n_0 \cdot \dots \cdot n_j$, i.e. the cardinality of the cross product of all results for bgp_0, \dots, bgp_j . It follows that:

$$n_0 \cdot \prod_{j=1}^i prop_j = sel_{bgp_0, \dots, bgp_j} \cdot \prod_{j=0}^i n_j \quad (5)$$

and consequently:

$$E = \sum_{i=1}^m \frac{fpr_i}{sel_{bgp_0, \dots, bgp_i} \cdot \prod_{j=0}^{i-1} n_j} . \quad (6)$$

Equation 6 shows that the higher the number of joins and the lower the join-selectivities $sel_{bgp_0, \dots, bgp_j}$ are, the smaller the false positive rate fpr_i of the approximation must be to produce a reasonably small estimation error. Thus, the *approximation error is determined by the **ratio between the false positive rate and selectivity*** and not “just” the false positive rate. In addition, this error does not only lead to inaccurate results but *also has a negative impact on the execution time* of the approximation: If the selectivities are low and the false-positive rates relatively high, it can happen that the query approximation mainly processes false-positives and that the data synopses based on these false-positives are larger than the actual data of all true-positives. Thus, the *query approximation might take longer than the actual query execution*.

Note that these theoretical findings should be cause for concern for building federated query systems in the light of false positive baring data synopses. In the next section, we will explore if these theoretical considerations apply to the practical Web of Data setting that we are currently exploring in federated querying.

Verification of the Analysis: We want to verify that our theoretical analysis indeed serves as an explanation of the error and runtime behavior that we observed in Section 3. For this, we compared the estimated error predicted by our analysis with the actual error which we observed in our evaluation. Figure 3 plots the estimated error based on equation 6 against the actual relative error measured for the Bloom Filter approximation in a log-log scale (as the values include both very small and very large numbers). Figure 3 suggests a very strong correlation between relative error of the estimation and the predicted error by our analysis. Indeed, both the Pearson correlation coefficient $R^2 = 0.81$ and the Spearman’s Rank Correlation $\rho = 0.76$ between the actual (non-log) numbers indicate a strong correlation between the theoretical estimation of the error and the actual evaluation. Not included in the figure, but included in the calculation of the correlation coefficients are those estimates that produced a relative error of 0, which could not be drawn in the log-log scale plot.

The figure shows that for a false positive probability of 10^{-8} (indicated by little pluses “+” mostly at the top left of the Figure) the actual error is not as small as one might expect. One likely explanation for this is the fact that the

specified false positive rate only gives a lower bound on the actual false positive rate, as we already discussed in Section 3.

Overall, Figure 3 confirms the theoretical analysis of the error accumulation in Equation 6, which indicates that *SPARQL queries require data synopses with very low false-positive rates to produce reasonably accurate results – an effect which is much more pronounced for queries with a low selectivity, as we have shown in Equation 6.* This, in turn, might require specific implementations of Bloom Filters that can handle such low probabilities. However, the need for such accurate synopses make it questionable whether join approximations that produce false positives are suitable for such tasks, in general.

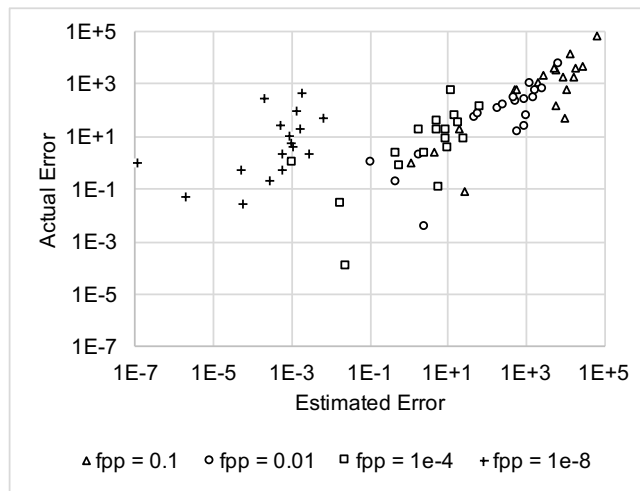


Fig. 3. Estimated error plotted against actual error.

5 Limitations and Future Work

In this paper, we focused our evaluations on queries which did not include UNION, OPTIONALs, and FILTERs outside of service patterns. However, we think, given the performance of our Bloom Filter approximation in this simpler setting, one cannot expect the approximations to perform better when extending the evaluation to include more complex queries. In particular, joins over multiple variables are likely to further constrain a join offering even more potential for synopses to generate false positives. UNIONS can be seen as a conjunction of multiple queries, which does not pose a significantly different setting. We will have to consider OPTIONALs in future work, though our intuition indicates that they can be seen as a combination of two different queries, which should not impact our results. FILTER expressions are more complex and warrant future work, as they might impact synopses construction. In particular, when a

filter compares two bindings from different sources it can only be applied after the join, which may require executing it on the actual data (rather than only on a synopsis), anyway.

Obviously, Bloom Filters represent only a one possible method to estimate the join-cardinality of SPARQL queries. Our theoretical considerations, however, are based on the fact that most synopses have false positives, so we do expect these findings to generalize.

Our assumption that each triple pattern must be sent to a different source results in a high number of joins between different endpoints. In practice, it could be that many queries may not have to be distributed to such an extent and subqueries with multiple triple patterns may be answered by a single endpoint. As the WoD grows, however, we are likely to see a rising number of queries that are getting bigger and are increasingly distributed. Hence, we believe that our findings do point to a core problem of federated querying on the Web of Data.

6 Conclusion

This paper set out to investigate the applicability of query approximation for source selection. We hypothesized that the performance of cardinality approximations of federated SPARQL queries degenerates when applied to queries with multiple joins of low selectivity. Indeed, both our empirical evaluation and our theoretical considerations indicate that data synopses are not suitable for this task due to their cumulative error, which also substantially slows down the estimation process. Based on our analysis, one can only expect good approximation performance if (1) the number of joins is low, (2) the join-selectivity is high, and/or (3) there is a bias which causes true positive matches to have a much higher propagation rate than false positive matches. These findings seriously hamper the usefulness of current selectivity estimation techniques for domains such as the WoD, where the number of joins involved in the estimation process is high. Indeed, our focus on a setting with many joins pinpointed a deficit in the generalizability of selectivity estimation techniques which came from a domain where usually only few inter-domain-joins are to be expected.

It is important to note that whilst this paper focused on federated SPARQL-querying in the context of the WoD our findings generalize to any federated conjunctive querying setting where join estimates cannot be precomputed.

The consequence of our work is twofold: First, to fulfil the Semantic Web vision via federated querying requires a *renewed effort to find suitable join-approximations for federated SPARQL queries*. As the WoD progresses, we will require more sophisticated approximation techniques, which are more adapted to the WoD: i.e., the need to be able to handle many inter-source joins and low selectivity better. Note, however, that no matter what new technique gets introduced, in the presence of low selectivity, our analysis of the error propagation adds a limit to what can be achieved by join-approximations that cause false-positives.

Second, if the community does not manage to drastically improve approximation techniques, *there might be a need to consolidate datasets from different sources into more centralized structures to reduce the number of endpoints that must be accessed during federated query execution.* This centralization will allow computing better estimations or incorporating them into the indices.

In conclusion, our findings showed that we may have to rethink well-known techniques such as the concept of join-approximation when applying them to the WoD. Doing so, will both advance our understanding of these techniques and may cause us to rethink the structure of the Web of Data as a whole.

Acknowledgments This work was partially supported by the Swiss National Science Foundation under grant #153598.

References

1. Acharya, S., Gibbons, P.B., Poosala, V., Ramaswamy, S.: Join synopses for approximate query answering. In: Proc. of the 1999 ACM SIGMOD International Conference on Management of Data. pp. 275–286 (1999)
2. Acosta, M., Vidal, M.E., Lampo, T., Castillo, J., Ruckhaus, E.: Anapsid: An adaptive query processing engine for sparql endpoints. In: International Semantic Web Conference (1). pp. 18–34 (2011)
3. Basca, C., Bernstein, A.: Querying a messy Web of Data with Avalanche. *Journal of Web Semantics* 26, 1–28 (2014)
4. Belson, D.: Akamai’s [state of the internet]. Q3 2015 report. Tech. rep., Akamai Technologies (2015)
5. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. In: Communications of the ACM. vol. 13, pp. 422–426 (1970)
6. Bose, P., Guo, H., Kranakis, E., Maheshwari, A., Morin, P., Morrison, J., Smid, M., Tang, Y.: On the false-positive rate of Bloom Filters. *Information Processing Letters* 108(4), 210–213 (2008)
7. Buil-Aranda, C., Arenas, M., Corcho, O., Polleres, A.: Federating queries in sparql 1.1: Syntax, semantics and evaluation. *Web Semantics: Science, Services and Agents on the World Wide Web* 18(1), 1–17 (2013)
8. Chakrabarti, K., Garofalakis, M., Rastogi, R., Shim, K.: Approximate query processing using Wavelets. In: The VLDB Journal. vol. 10, pp. 199–223 (2001)
9. Görlitz, O., Staab, S.: SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In: Proceedings of the 2nd International Workshop on Consuming Linked Data. vol. 782, pp. 13–24. Bonn, Germany (2011), http://uni-koblenz.de/~goerlitz/publications/GoerlitzAndStaab_COLD2011.pdf
10. Harris, S., Seaborne, A.: SPARQL 1.1 query language. <https://www.w3.org/TR/sparql11-query/> (March 2013)
11. Hose, K., Schenkel, R.: Towards benefit-based RDF source selection for SPARQL queries. In: SWIM ’12 Proceedings of the 4th International Workshop on Semantic Web Information Management. Scottsdale, Arizona (2012)
12. Ioannidis, Y.E., Christodoulakis, S.: On the propagation of errors in the size of join results. In: Proceedings of ACM SIGMOD Conference. pp. 268–277 (1991)
13. Ioannidis, Y.E., Poosala, V.: Histogram-based approximation of set-valued query answers. In: Proceedings of the 25th International Conference on Very Large Data Bases. pp. 174–185 (1999)

14. Kossmann, D.: The state of the art in distributed query processing. *ACM Computing Surveys* 32(4), 422–469 (2000)
15. Lipton, R.J., Naughton, J.F., Schneider, D.A.: Practical selectivity estimation through adaptive sampling. In: *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*. pp. 1–11 (1990)
16. Moor, D., Grubenmann, T., Seuken, S., Bernstein, A.: A double auction for querying the web of data. In: *The Third Conference on Auctions, Market Mechanisms and Their Applications* (2015)
17. Moor, D., Seuken, S., Grubenmann, T., Bernstein, A.: Core-selecting payment rules for combinatorial auctions with uncertain availability of goods. In: *Twenty-Fifth International Joint Conference on Artificial Intelligence*. pp. 424 – 432 (2016)
18. Ozsu, T., Valduriez, P.: *Principles of Distributed Database Systems* (2nd Edition). Prentice Hall, 2 edn. (1999), <http://www.citeulike.org/user/zflavio/article/379597>
19. Prasser, F., Kemper, A., Kuhn, K.A.: Efficient distributed query processing for autonomous RDF databases. In: *Proceedings of the 15th International Conference on Extending Database Technology - EDBT '12*. pp. 372–383 (2012)
20. Quilitz, B., Leser, U.: Querying distributed RDF data sources with SPARQL. *Proceedings of the 5th European semantic web conference on The semantic web: research and applications* pp. 524–538 (2008)
21. Saleem, M., Ngonga Ngomo, A.C.: HiBISCuS: Hypergraph-based source selection for sparql endpoint federation. In: *The Semantic Web: Trends and Challenges*. pp. 176–191 (2014)
22. Saleem, M., Ngonga Ngomo, A.C., Parreira, J.X., Deus, H.F., Hauswirth, M.: DAW: Duplicate-Aware federated query processing over the Web of Data. In: et al., A.H. (ed.) *The Semantic Web – ISWC 2013. Lecture Notes in Computer Science*, vol. 8218, pp. 574–590. Springer, Berlin, Heidelberg (2013)
23. Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: FedBench: A benchmark suite for federated semantic data query processing. *International Semantic Web Conference* pp. 585–600 (2011)
24. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: Optimization techniques for federated query processing on Linked Data. In: *Proceedings of the 10th International Semantic Web Conference*. pp. 601–616 (2011)
25. Sheth, A.P., Larson, J.A.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys* 22(3), 183–236 (Sep 1990)
26. Umbrich, J., Hose, K., Karnstedt, M., Harth, A., Polleres, A.: Comparing data summaries for processing live queries over Linked Data. *World Wide Web* 14(5), 495–544 (2011)
27. Vidal, M.E., Castillo, S., Acosta, M.: On the selection of SPARQL endpoints to efficiently execute federated SPARQL queries. In: *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXVII*. pp. 109–149 (2016)