Towards Efficient Query Processing over Heterogeneous RDF Interfaces*

Gabriela Montoya[1], Christian Aebeloe[1], and Katja Hose[1]

Aalborg University, Denmark
{gmontoya,caebel,khose}@cs.aau.dk

Abstract. Since the proposal of RDF as a standard for representing statements about entities, diverse interfaces to publish and strategies to query RDF data have been proposed. Although some recent proposals are aware of the advantages and disadvantages of state-of-the-art approaches, no work has yet tried to integrate them into a hybrid system that exploits their, in many cases, complementary strengths to process queries more efficiently than each of these approaches could do individually. In this paper, we present HYBRIDSE, an approach that exploits the diverse characteristics of queryable RDF interfaces to efficiently process SPARQL queries. We present a brief study of the characteristics of some of the most popular RDF interfaces (brTPF and SPARQL endpoints), a method to estimate the impact of using a particular interface on query evaluation, and a method to use multiple interfaces to efficiently process a query. Our experiments, using a well-known benchmark dataset and a large number of queries, with result sizes varying from 1 up to 1 million, show that HYBRIDSE processes queries up to three orders of magnitude faster and transfers up to four orders of magnitude less data.

1 Introduction

Efficiently evaluating SPARQL queries over heterogeneous RDF interfaces, such as SPARQL endpoints and Triple Pattern Fragments (TPFs) [26], requires considering their individual characteristics through all steps of query optimization and query processing. For example, some interfaces exhibit their best performance for answering basic queries without joins (triple pattern queries) while other interfaces exhibit their best performance for answering queries with multiple joins (BGP queries). However, answering SPARQL queries is expensive in general. Even if only join, filter, and union operators are considered, deciding whether a result mapping is an answer to a query is already NP-complete [21]. A well-known heuristic for executing joins is reducing the sizes of intermediate results, in particular when they have to be transferred between clients and servers. As in [13], this paper focuses on BGP queries to ease the presentation of the proposed approach and provide insightful empirical

* This work will be published as part of the book "Emerging Topics in Semantic Technologies. ISWC 2018 Satellite Events. E. Demidova, A.J. Zaveri, E. Simperl (Eds.), ISBN: 978-3-89838-736-1, 2018, AKA Verlag Berlin".

results. However, our approach and the gained insights are also applicable to more complex fragments of the SPARQL query language.

In this paper, we examine and show how the advantages of available queryable RDF interfaces can be exploited to find an execution plan that makes the best use of the strengths of available interfaces. We propose HYBRIDSE (**hybrid** **S**PARQL **E**ngine), a smart client that is aware of alternative interfaces for the same dataset, exploits their strengths to evaluate different parts of a query and combines partial answers to produce the final result.

In summary, this paper makes the following contributions:

- Characterization of existing queryable RDF interfaces: SPARQL endpoints, triple pattern fragments, and bindings-restricted triple pattern fragments
- Estimation of the impact of different RDF interfaces on query execution
- Method to compute good execution plans exploiting advantages of different RDF interfaces
- Extensive evaluation using the well-known WatDiv benchmark [2]

This paper is organized as follows. Section 2 presents related work, Section 3 characterizes different RDF interfaces, Section 4 discusses the impact of RDF interfaces on query execution, Section 5 sketches HYBRIDSE, Section 6 discusses our experimental results, and finally Section 7 concludes the paper.

2 Related Work

The basic idea of exploiting multiple data sources and interfaces has been considered in the database community in the context of multistore systems or polystores [4]. These systems enable querying different systems (RDBMS, NoSQL, HDFS, etc.) depending on what would be the most efficient paradigm for a particular query.

As with other data models, SPARQL queries can be evaluated using different interfaces, such as SPARQL endpoints, link traversal over RDF documents [14], Triple Pattern Fragments [26], or bindings-restricted Triple Pattern Fragments [13]. These interfaces, in combination with different fragments of the SPARQL language, represent restrictions on expressiveness. They have different purposes and strengths, and the most optimal interface can vary depending on the specific characteristics of a particular query and the user's needs.

Recently, Hartig et al. [15] have explored the expressiveness of using different LDF interfaces in combination with different clients from a theoretical point of view. Even though this work mentions the advantages of using some interfaces over others for a given class of queries and regarding certain metrics, its practical use remains limited. For instance, although using a SPARQL endpoint to evaluate a complete query is the best in terms of the number of requests and the amount of transferred data, having multiple clients choosing to evaluate their queries using a particular interface can decrease the query response time and deteriorate the user satisfaction. To that end, using an approach like multistores in the context of the Semantic Web could be beneficial in delivering the fastest query execution plan. Although multistores provide a more diverse

query interface, HYBRIDSE still relies on the same data model (RDF) but focuses on stores with different and in many cases complementary strengths.

Efficiently querying multiple homogeneous interfaces that provide access to different datasets, i.e., processing federated SPARQL queries, has been extensively studied, in the context of SPARQL endpoints [1, 3, 7, 11, 12, 17–19, 22, 23, 25] and LDF [26]. Some of these works exploit knowledge about the data available through the interfaces, for instance, estimating the cardinality of joining data available through different interfaces [18] or pruning sources that only provide redundant data [19]. In contrast to these techniques, the work presented in this paper considers heterogeneous interfaces providing access to the same dataset. However, some of the query optimization strategies developed in the contexted of federated SPARQL queries could be combined with our approach. For instance, in combination with the optimizations proposed in [19], the heterogeneous interfaces may provide access to different, possibly overlapping, fragments of datasets.

3 Characteristics of RDF Interfaces

In this section, we briefly discuss the main characteristics of some of the most popular queryable RDF interfaces.

3.1 SPARQL Endpoints

SPARQL endpoints are specifically designed to efficiently query RDF datasets. Having access to the whole dataset in advance allows for computing indexes and enables the use of tailored physical operators according to the characteristics of the data. Therefore, SPARQL endpoints can, for instance, rely on cost functions to determine whether a symmetric hash join is more suitable than a bound join to evaluate a join in a given query.

However, being able to evaluate any (or almost any) SPARQL query comes at a cost. Query optimization in consideration of all available alternatives (indexes, auxiliary structures, join order, join algorithm, etc.) can represent an unnecessary overhead for relatively simple queries, for which a straightforward optimization without considering alternatives would result in a sufficiently good query execution plan. Moreover, in some cases cost functions rely on low quality cardinality estimations and lead to produce execution plans with very large execution times [9].

3.2 Triple Pattern Fragment (TPF)

Verborgh et al. [26] proposed the Triple Pattern Fragment (TPF) interface to reduce query execution costs and distribute costs among server and clients. Using TPF, the server is only responsible to provide answers to triple pattern queries, i.e., queries composed of a single triple pattern, and the client is responsible for processing operators such as join, union, and optional. Due to the reduction of the load at the servers, these interfaces allow for a better scalability of services providing access to RDF sources.

3.3 Bindings-Restricted Triple Pattern Fragments (brTPF)

Even if the costs of providing access to RDF data is considerably reduced by using TPF interfaces, the paradigm used by TPF leads to less efficient query executions [13]. In particular, the fact that the evaluation of joins relies on the evaluation of triple patterns by the server and the subsequent construction of new triple pattern queries with the bindings obtained from the previous triple pattern evaluation, leads to a large amount of data transferred between server and clients (in both directions). Therefore, Hartig and Buil Aranda [13] proposed an extension of the TPF interface that allows to include, in addition to a triple pattern, bindings in the queries sent to the server. This extension allows for a significant reduction in the number of requests and the amount of data transferred from the server to the clients, without decreasing the query throughput [13].

3.4 Overview

Table 1 shows an overview of the studied interfaces. For powerful servers and low numbers of clients, SPARQL endpoints represent the best option to efficiently process queries as they transfer the lowest amount of data and have no requirement on the client side resources. For very high numbers of clients with no requirements regarding answer time and some local resources available for query processing, the TPF interface offers the best option. A server can respond to a very large number of simple queries while the clients take over most of the query processing tasks. Lastly, if clients and servers have resources available for query processing, brTPF is the best option. As computational power spent by brTPF servers to process requests is between SPARQL endpoints' and TPF servers', brTPF servers can answer more requests. Moreover, as these requests reduce the network traffic, query results are obtained faster.

Table 1: RDF Interfaces Overview: characteristic resource usage from the **server**, **network** (transferred data), and **client**

	Server	Network	Client
SPARQL Endpoint	Indexes, cost functions loop and hash join	**results**	-
TPF	-	Intermediate results	Pipelined nested loop join
brTPF	Bind join support	Intermediate results	Pipelined **bind** nested loop join

3.5 Assumptions

In this paper, we assume that the most expressive interface, i.e., SPARQL endpoints, takes advantage of having access to server-side resources to efficiently process queries by relying on indexes. We further assume that the less expressive interfaces, i.e., TPF [13], are used in combination with back-ends that provide fast access to triple pattern fragments and good estimations of the number of triples (cardinality) matching a triple pattern. While these assumptions clearly

impose some restrictions about the implementation of the RDF interfaces, they hold for commonly used implementations, such as Virtuoso endpoints and TPF with HDT backends [6]. Validating our conclusions using unconventional implementations is out of the scope of this paper.

4 Interface Impact on Query Evaluation Efficiency and Resource Usage

Query processing time mainly depends on three aspects: (i) query processing time at the server, (ii) transfer time of (intermediate) results from servers to clients, and (iii) query processing time at the client. While all aspects are influenced by the size of intermediate results, aspects (i) and (iii) are also influenced by the data structures used to store intermediate results during query processing, e.g., available indexes and the complexity of the algorithms used to process the query.

A restricted interface, such as TPF, has very low query processing time at the server, transfer time depends on the sizes of intermediate results, and query processing time at the client can be significantly high depending on the selectivity of the triple patterns in the query. Moreover, queries with high numbers of intermediate results lead to a high number of requests to the TPF server during query processing, which can significantly slow down execution.

A SPARQL endpoint has a significantly higher query processing time at the server, transfer time depends on the query result size, and query processing time at the client is very low (and usually negligible).

If the SPARQL endpoint had enough query processing power to instantly answer the queries of all users, then the best possible approach to efficiently process queries would be to use the SPARQL endpoint in all cases. However, the available query processing power of SPARQL endpoints is limited and this processing power should be used mainly in cases when using other interfaces would incur in significantly higher query processing times in order to improve the overall efficiency of query processing for all clients.

Knowing the sizes of intermediate and query results could help deciding which interface to use to execute the queries. While computing this knowledge before execution time has been addressed in diverse works [5, 8, 10, 18, 25], HYBRIDSE uses a simpler and more straightforward approach that relies on available metadata provided by the TPF servers and heuristics. It therefore does not entail any additional computational costs. In particular, we can use the cardinality of a triple pattern fragment, i.e., number of triples in the triple pattern fragment, as a generally good enough estimate is provided by TPF servers with HDT backend, whenever a triple pattern fragment is retrieved. While processing a query using a brTPF client, fragments and their cardinality estimation, are retrieved. We propose to use the fragment cardinality estimation to decide if continuing using brTPF is a good choice, or if using the SPARQL endpoint is likely to be more efficient. If the estimated fragment cardinality is low, then using a very simple plan, produced by the brTPF client, and retrieving the data from the brTPF server is usually a good enough approach. However, if

the estimated fragment cardinality is high, then exploiting existing information and structures used by SPARQL endpoints will most likely represent a significant advantage in terms of query execution time. We use a threshold value to assess if the fragment cardinality is low or high.

To illustrate the impact of the interfaces on query efficiency and resource usage, we use queries generated using the WatDiv benchmark [2] query generator and a 10 million triples dataset. Example queries A and B are presented in Listings 1.1 and 1.2. Table 2a shows the estimated fragment cardinalities of the triple patterns in these queries and Table 2b shows the query execution times and numbers of transferred bytes from the brTPF server and the Virtuoso endpoint (version 7.2.4.2); these queries were run in setups with 4 and 16 clients.[1]

Listing 1.1: Query A: Find purchases of friends of reviewers who reviewed products liked by the users followed by "User1204"

```
SELECT * WHERE {
 wsdbm:User1204 wsdbm:follows ?v1 . (tp1)
 ?v1 wsdbm:likes ?v2 .              (tp2)
 ?v2 rev:hasReview ?v3 .            (tp3)
 ?v3 rev:reviewer ?v4 .             (tp4)
 ?v4 wsdbm:friendOf ?v5 .           (tp5)
 ?v5 wsdbm:makesPurchase ?v6        (tp6)
}
```

Listing 1.2: Query B: For products that can be delivered in country "Country4", output their prices and information about their retailers

```
SELECT * WHERE {
 ?v1 schema:eligibleRegion wsdbm:Country4 . (tp1)
 ?v0 goodrelations:offers ?v1 .             (tp2)
 ?v1 goodrelations:price ?v2 .              (tp3)
 ?v0 schema:contactPoint ?v3 .              (tp4)
 ?v0 schema:email ?v5 .                     (tp5)
 ?v0 schema:legalName ?v6 .                 (tp6)
 ?v0 schema:openingHours ?v7 .              (tp7)
 ?v0 schema:paymentAccepted ?v8             (tp8)
}
```

Query A (Listing 1.1) has one small fragment, tp1, and joins on different variables. Thus, Query A is evaluated more efficiently by brTPF, which results in a relatively low amount of data transfer in comparison to the number of results, while the endpoint relies on cardinality estimations that are deteriorated by the different object-subject joins and the unsatisfied assumption of query triple pattern independence [20].

Query B (Listing1.2) involves only fragments with relatively large estimated cardinalities and many subject-subject joins on the same variables. Then, using brTPF results in relatively high amounts of data transfer in comparison to

[1] In these setups, described in Section 6, each client executes around 200 different queries. Queries A and B are two example queries executed by one of the clients.

Table 2: Estimated fragment cardinality, number of results (NR), and evaluation metrics execution time (ET) and number of transferred bytes (NB) for Queries A and B

(a) Estimated fragment cardinalities

Query	tp1	tp2	tp3	tp4	tp5	tp6	tp7	tp8
Query A	**47**	23,921	5,159	150,000	39,781	15,829		
Query B	10,495	1,199	240,000	953	91,004	**108**	962	703

(b) Evaluation metrics

Query	4 clients				16 clients			
	Endpoint		brTPF		Endpoint		brTPF	
	ET	NB	ET	NB	ET	NB	ET	NB
Query A	139,676	10,178,792	47,536	12,094,040	175,734	10,178,792	95,674	12,094,040
Query B	241	227,338	36,237	3,820,006	359	227,338	286,812	3,820,006

the number of results. Virtuoso is able to exploit existing indexes to efficiently process Query B outperforming brTPF for this particular query.

As the number of calls to the servers increases with the number of clients issuing queries, the servers represent bottlenecks, which can result in higher response times.

5 Efficient Query Execution over Available Interfaces

Algorithm 1 Evaluate BGP

Input: sorted triple patterns *tps* in the BGP; set of initial mappings *ms*; *threshold* that guides the choice of interface
Output: A set of answer mappings (*ms*) to BGP *tps*

```
1: function evaluateBGP(tps,ms,threshold)
2:     if size(tps)=0 then
3:         return ms
4:     end if
5:     if estimatedFragmentCardinality(first(tps)) > threshold ∧ size(tps) > 1 then
6:         ms' ← evaluateBGPAtSPARQLEndpoint(tps, ms)
7:         ms ← ms' ⋈ ms
8:     else
9:         first ← first(tps)
10:        tps ← removeFirst(tps)
11:        ms' ← evaluateTPAtBrTPFServer(first, ms)
12:        ms ← ms' ⋈ ms
13:        ms ← evaluateBGP(tps, ms,threshold)
14:    end if
15:    return ms
16: end function
```

The goal of HYBRIDSE is to use a combination of multiple interfaces to reduce the overall query execution time. The simpler interface should be used

for relatively simple computations, and if we already know there are not many intermediate results, then an interface like brTPF is the best. However, in some cases applying the simpler query execution approach and using the simpler interface leads to very expensive query execution times. In such cases, we should make use of more costly interfaces that are able to exploit knowledge about the data to produce better execution plans.

Algorithm 1 sketches how HYBRIDSE implements the aforementioned heuristics. The list of triple patterns and their order is described in *tps*. Hence, to achieve efficient query processing, *tps* should be sorted according to selectivity and this can be based on several heuristics. The order used in this paper is as follows: The first triple pattern is the one with the lowest estimated fragment cardinality. The second triple pattern is the one with the greater number of bound variables, when considering the bound variables after the evaluation of the first triple pattern, and so on. As such, it is more likely that fragment cardinalities will be reduced as the number of bound variables increases. Note that it is possible to use any other execution order for triple patterns, e.g., updating the order of the triple patterns after the evaluation of each triple pattern query at the brTPF server.

For Query A (Listing1.1), the first triple pattern is *tp1* with an estimated fragment cardinality of 47. The next one to execute is *tp2* because after evaluation of *tp1*, we know the bindings for variable *?v1* and can use them to obtain a smaller fragment for triple pattern *tp2*.

The estimated fragment cardinality of the triple pattern with higher selectivity is used to determine whether brTPF should be used or if it is necessary to use a more expressive interface, i.e., the SPARQL endpoint (Line 5), as we consider that the triple patterns in *tps* are sorted according their selectivity this is the first triple pattern in *tps*. In any case, the previously obtained mappings are passed on to the function evaluating the graph pattern (BGP or TP), line 6 or 11, and only the mapping values that are relevant for the pattern evaluation are included in the request sent to the server.

Listing 1.3: Query C: Find the purchases of friends of the reviewers of a set of products

```
SELECT * WHERE {
 VALUES (?v2 ) {
  (wsdbm:Product24957) (wsdbm:Product20003)
  (wsdbm:Product14391) (wsdbm:Product18039)
  (wsdbm:Product19333) (wsdbm:Product18687)
  (wsdbm:Product10415) (wsdbm:Product13683)
  (wsdbm:Product15505) (wsdbm:Product202)
  (wsdbm:Product1262)
 }
 ?v2 rev:hasReview ?v3.
 ?v3 rev:reviewer ?v4.
 ?v4 wsdbm:friendOf ?v5.
 ?v5 wsdbm:makesPurchase ?v6.
}
```

For Query A (Listing1.1) the first triple pattern (*tp1*) is evaluated first and the 46 values of *?v1* are used to evaluate *tp2* using the brTPF interface; the 46 values are used to build two brTPF requests, one with 30 bindings and another one with 16 bindings, considering a maximum of 30 bindings per request. As the resulting fragments have estimated cardinalities of 26 and 15, the evaluation continues using brTPF. The 41 values obtained for *?v2* are used to build two brTPF requests to evaluate *tp3*. As the retrieved fragments have estimated fragment cardinalities of 489 and 113, thresholds of up to 112 result in evaluating the triple patterns { tp3 . tp4 . tp5 . tp6 } using the Virtuoso endpoint. As such, the values for *?v2* are passed on to the Virtuoso endpoint in a VALUES clause, as shown in Listing 1.3, rather than to the brTPF server. Notice that the difference between the estimated fragment cardinality, 15, and the actual fragment cardinality, 11, used to generated Query C, cf. Listing 1.3, is relatively low. Likewise, the obtained 30 bindings from the other brTPF request, also close to the estimated fragment cardinality of 26, are used to build a similar request for the Virtuoso endpoint.

Table 3 shows the evaluation metrics for HYBRIDSE obtained when executing queries A and B (Listings 1.1 and 1.2) in the setups with 4 and 16 clients introduced in Section 4, and used to evaluate brTPF and endpoint (values in Table 2b). While Query A has been evaluated as described above exploiting the advantages of both brTPF and endpoint, the execution of Query B relies exclusively on the endpoint. Using HYBRIDSE, Query A is executed considerably more efficient, at least two times faster for the setup with 4 clients and 6 times faster for the setup with 16 clients, and achieves a similar performance as a SPARQL endpoint for Query B.

Table 3: Execution time (ET) and number of transferred bytes (NB) for Queries A and B using HYBRIDSE

Query	4 clients			16 clients		
	ET	**NB** Endpoint	**NB** brTPF	**ET**	**NB** Endpoint	**NB** brTPF
Query A	20,191	10,665,981	21,103	15,362	10,665,981	21,103
Query B	333	233,478	0	547	233,478	0

6 Evaluation

To evidence the potential of HYBRIDSE, we have implemented a prototype to evaluate BGP queries using a combination of SPARQL endpoints and brTPF servers. The prototype uses Algorithm 1 described in Section 5 to exploit the advantages of each RDF interface.

Dataset and queries: We use the WatDiv benchmark [2] query generator and a generated 10 million triples dataset to conduct an experiment with multiple clients – similarly as done in [13]. To show that HYBRIDSE works well with a wide range of queries, we use queries with different result sizes[2], i.e., queries with result sizes in the ranges (1,10^2], (10^2, 10^3], (10^3, 10^4], (10^4, 10^5], and (10^5, 10^6]. We consider up to 32 clients, each client having around 200 different queries [3] to execute, i.e., in total around 6,400 queries are executed in the setup with 32 clients. While clients run in parallel, each client executes its queries sequentially, i.e., in a given instant, there are up to 32 queries executed in parallel.

Implementation: The SPARQL endpoint was deployed using Virtuoso 7.2.4.2. We used the brTPF client and server provided by its authors [13][4]. HYBRIDSE is implemented in Node.js on top of the brTPF client implementation and it is available on our website http://qweb.cs.aau.dk/hybridse.

Hardware configuration: For our experiments we used virtual machines (VMs) running Ubuntu Linux 4.4.0 64bit x86 with eight cores and CPU 2294.250 MHz with a network speed of up to 10,000Mb/s. A dedicated VM was used to deploy the servers. A Virtuoso 7.2.4.2 endpoint was set up to use up to 7GB of RAM, 1 million result set rows, 6,000s estimated execution time, and 600s execution time. A brTPF server was set up to use up to 8GB of RAM. Up to four VMs, with 16GB of RAM, were used to run clients for the experiments. Each (client) VM ran up to eight clients simultaneously.

Evaluation metrics:

1. *Execution Time (ET)* is the time elapsed since the query is issued until the complete answer is produced (with a timeout of 300,000 ms),

[2] Queries are available on our website http://qweb.cs.aau.dk/hybridse

[3] The actual number of queries per client is between 198 and 203. Some clients have less queries because the generator failed to produce enough different queries within the largest result size range.

[4] As available at http://olafhartig.de/brTPF-ODBASE2016/ on January 2018. These implementations have some limitations, as pointed out by Hartig in his review of our work https://openreview.net/forum?id=BJeDc51elX, the client is purposely simple and lacks adaptive query optimization techniques to exploit metadata obtained during query processing and the cardinality estimations provided by the server *can be totally inaccurate*.

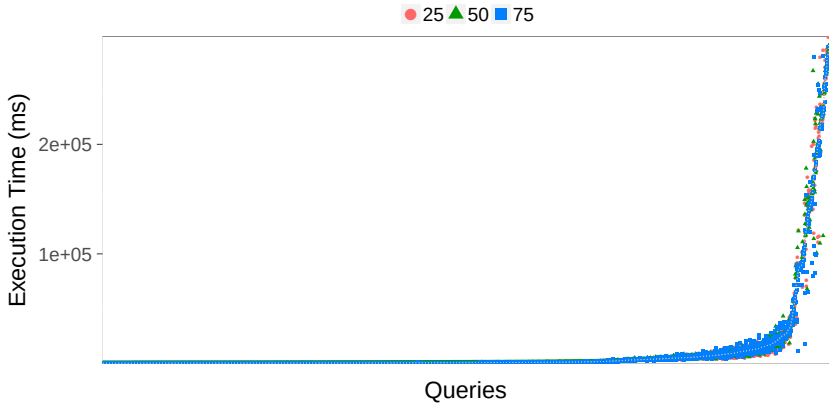
2. *Number of Transferred Bytes (NTB)* is the amount of data transferred from the brTPF server or the SPARQL endpoint to the query engine during query evaluation. It includes the Total Number of Transferred Triples from the brTPF server (TNTTTPF) and the Number of (sub)query Results obtained from the SPARQL Endpoint (NRSE). NTB is computed as the sum of bytes in the string representations of TNTTTPF and NRSE.
3. *Number of Calls to the brTPF server (NCTPF)* is the number of (sub)queries sent to the brTPF server.
4. *Number of Calls to the SPARQL Endpoint (NCSE)* is the number of (sub)queries sent to the SPARQL endpoint.
5. *Throughput (T):* is the number of queries executed by all the clients in one time unit. It is computed as the number of queries executed by the clients divided by the total execution time.

6.1 Experimental Results

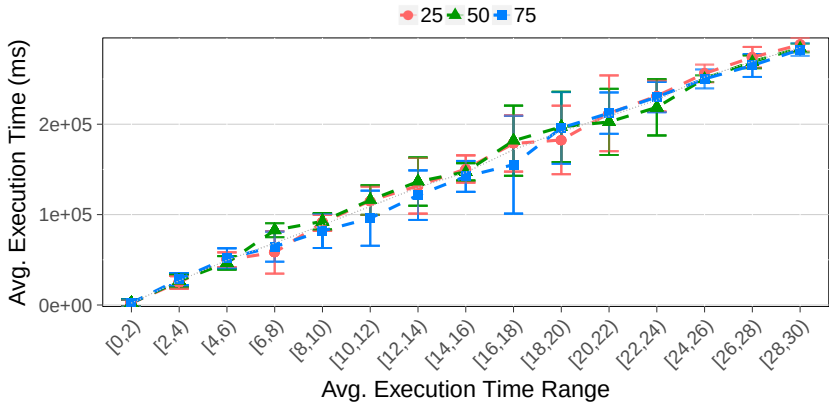
To evaluate the performance of HYBRIDSE, we compare its results to the SPARQL endpoint and the brTPF client. We compare the performance of the WatDiv queries based on the aforementioned metrics. HYBRIDSE has as input a threshold value representing the turning point where the least expressive interface should be switch to the most expressive one. In Algorithm 1, a threshold is used to choose which interface should be used to avoid inefficient processing of the queries from all the clients. We tested the WatDiv queries with three different thresholds: 25, 50, and 75. We also performed the WatDiv experiments with different numbers of clients: 1, 4, 8, 16 and 32. However, due to space limitations, we will only show results for 4 and 16 clients. After a comparison of HYBRIDSE's performance for the different thresholds, we will consider only threshold=50 for the rest of the paper. Timed out queries are omitted from the plots[5].

Because our evaluation comprises a large number of queries and to ease the readability of the results, we present the average values of the metrics for groups of queries with similar amounts of transferred bytes (NTB). As each approach might exhibit different NTB, we have considered for each query the average NTB, ANTB, over the three approaches, and divided the queries into clusters according to ANTB, such that the results obtained for the three approaches for each query are considered in the same cluster. Therefore, the comparison over these clusters corresponds to comparing the approaches for the same set of queries and eases the readability, i.e, in Figure 1b, instead of showing around 3,200 points per approach (one for each query), as in Figure 1a, we depict only 15 points (one per cluster), where each point represents the average value for queries with ANTB in the interval [x*10^6,y*10^6) for the label [x,y) in the x-axis of Figure 1b. Detailed results for all the setups and metrics are available at our website http://qweb.cs.aau.dk/hybridse.

[5] Fewer queries timed out using HYBRIDSE, e.g., in the 16 clients setup, only 134 out of 3,245 queries timed out, while 812 and 350 timed out using brTPF and endpoint.



(a) Detailed runtime per query



(b) Average runtime per cluster of queries

Fig. 1: Execution Time in ms (ET), 16 clients setup (x-axis, $*10^4$)

Impact of the threshold on hybridSE's performance Figure 1b shows the average execution time obtained when using different values of threshold in the setup with 16 clients. We observe that as the threshold increases, the performance of HYBRIDSE improves in average. The difference is quite small, having a very close average over all the queries with just slightly worse performance for rare queries when the threshold is 25.

Scalability of hybridSE Figure 2 shows the average execution time obtained when using different values for the threshold in a setup with 4 clients, where around 800 queries were executed. In comparison to Figure 1b, where around 3,200 queries were executed, we observe that even if the workload is multiplied by four, the execution time increases only sub-linearly.

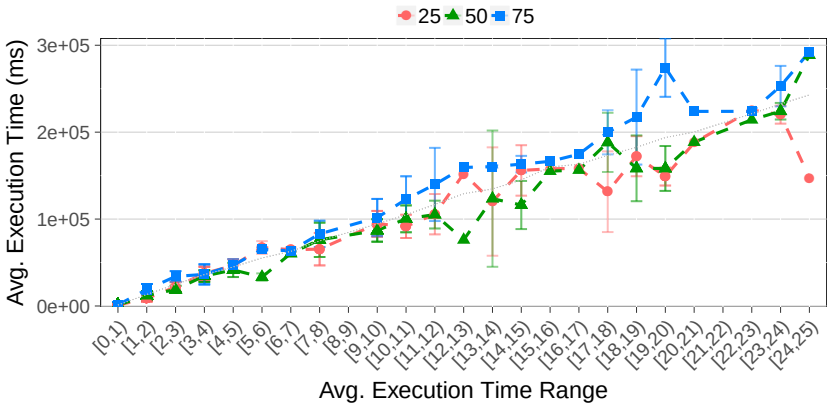


Fig. 2: Average Execution Time in ms (ET), 4 clients setup(x-axis, $*10^4$)

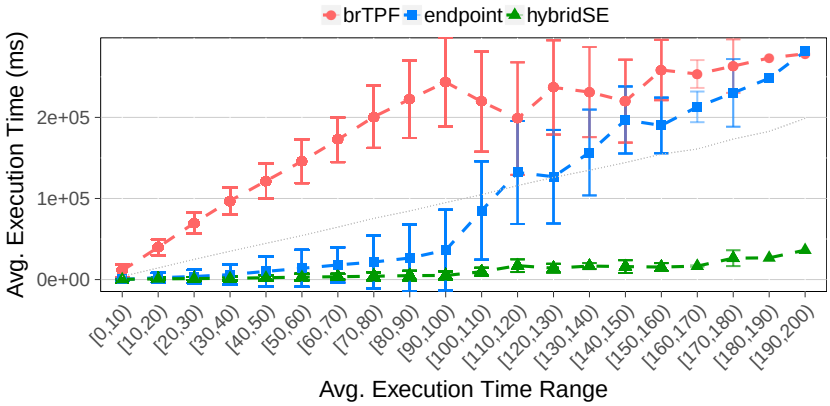


Fig. 3: Average Execution Time in ms (ET), 16 clients setup for BrTPF, Endpoint, and HYBRIDSE (x-axis, $*10^3$)

Execution time Figure 3 shows the average execution time obtained for HYBRIDSE, brTPF and endpoint in the setup with 16 clients. Even if brTPF and Endpoint faced challenging queries that have considerably degraded their execution time for some groups of queries, HYBRIDSE has exploited the advantages of each of these approaches, i.e., relying on brTPF only for very simple subqueries with one triple pattern or accessing relatively small triple pattern fragments that contribute to reducing the difficulty of the queries that are later sent to the SPARQL endpoint. These observations also hold for other setups with different number of clients and using different values of threshold.

Number of Transferred Bytes Figure 4 shows the average number of transferred bytes for HYBRIDSE and the brTPF and endpoint baselines in the setup with 16 clients. Clearly the endpoint baseline transfers the lowest number of bytes by transferring only the query results, and the brTPF baseline transfers higher numbers of bytes because the used interface only evaluates triple

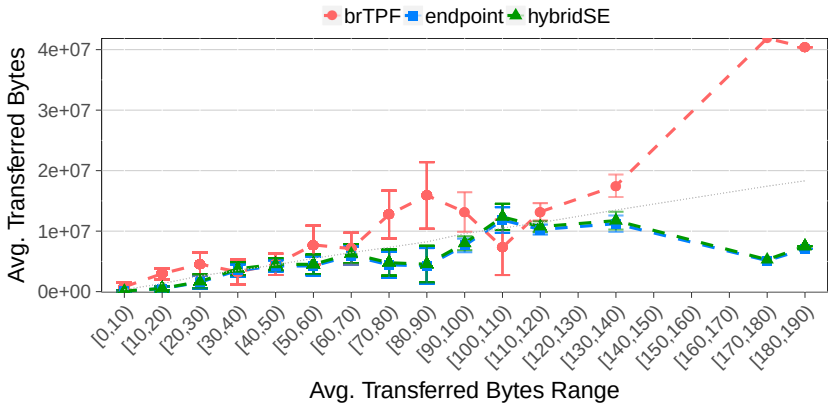


Fig. 4: Total Number of Transferred Bytes from the TPF server and the SPARQL Endpoint (NTB), 16 clients setup for BrTPF, Endpoint and HYBRIDSE approaches (x-axis, $*10^5$)

pattern queries with some binding values. HYBRIDSE remains very close to the data transfer incurred by the more efficient approach in this metric (endpoint), therefore using HYBRIDSE does not result in any considerable increase of the network traffic with respect to the best possible alternative.

Number of Calls Figure 5 shows the average number of calls to the TPF server and to the SPARQL endpoint for HYBRIDSE and the brTPF and endpoint baselines in the setup with 16 clients. Even if HYBRIDSE heavily relies on the SPARQL endpoint that can provide better performance for executing the queries, for the queries with triple patterns with estimated fragment cardinalities inferior to the threshold, it relies on the brTPF server instead. This allows for a better use of the available resources, using simpler interfaces whenever the performance of the evaluated queries is not likely to be severely impacted, i.e, when using brTPF results in relatively low number of calls.

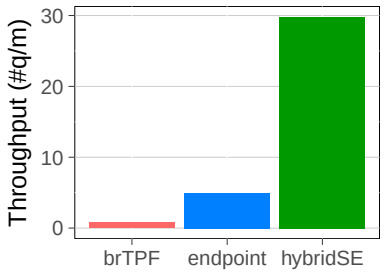
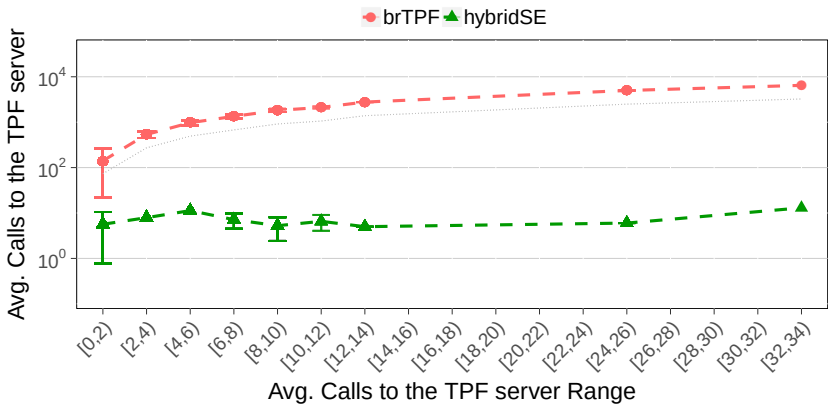
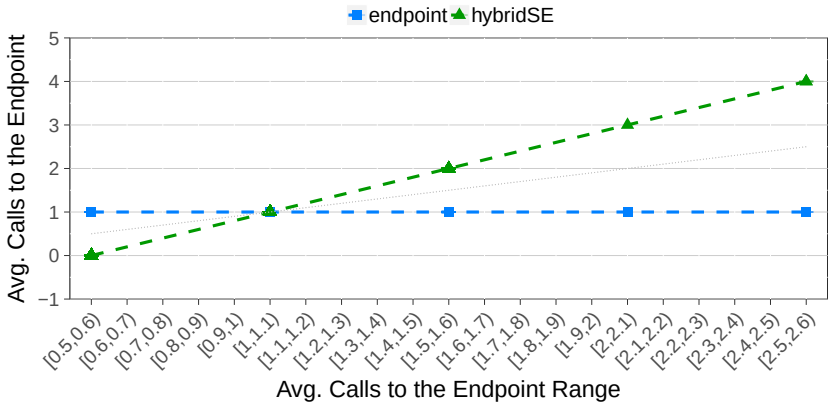


Fig. 6: System throughput, 16 clients

System Throughput Figure 6 shows system throughput for HYBRIDSE and the brTPF and endpoint baselines in the setup with 16 clients. The throughput of HYBRIDSE is considerably higher than the other approaches. HYBRIDSE is at least five times faster and up to 40 times faster than the baselines. Therefore, the improvement is not only due to having more resources available (two interfaces instead of one) but also due to making better use of these resources.



(a) Total Number of Calls to the brTPF server



(b) Number of Calls to the SPARQL Endpoint

Fig. 5: Total Number of Calls to the TPF server (TNCTPF, x-axis, $*10^2$) and to the SPARQL Endpoint (NCSE), 16 clients setup for BrTPF, Endpoint and HYBRIDSE approaches

7 Conclusion and Future Work

In this paper, we have presented HYBRIDSE, an approach for exploiting the different capabilities of available LDF interfaces to process queries as efficiently as possible but without unnecessary usage of resources. HYBRIDSE uses available information, such as the number of triples per fragment, to determine when it is more efficient to use the brTPF client or send a (sub)query to the SPARQL endpoint. Our experimental evaluation shows that HYBRIDSE produces query execution plans that are better in terms of data transfer and execution time than the available implementation for the brTPF interface.

As future work, it would be interesting to assess if the limitations of the available brTPF implementation had a significant impact on our experimental

results and to extend HYBRIDSE for more complex fragments of the SPARQL query language. Additionally, we would like to combine HYBRIDSE with other interfaces, such as other TPF clients, e.g., [16]. Finally, we would like to integrate our techniques into the Comunica platform [24]. Comunica allows to integrate different approaches to process queries over a combination of diverse RDF interfaces, however the diverse RDF interfaces are not yet exploited as alternative interfaces for the same dataset. Therefore, integrating HYBRIDSE into Comunica will facilitate the use of diverse RDF interfaces that access the same datasets and the use of the most recent implementations of the query engines HYBRIDSE relies upon.

Acknowledgments. This research was partially funded by the Danish Council for Independent Research (DFF) under grant agreement no. DFF-4093-00301 and Aalborg University's Talent Management Programme.

References

1. M. Acosta, M. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints. In *ISWC'11*, pages 18–34, 2011.
2. G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee. Diversified Stress Testing of RDF Data Management Systems. In *ISWC 2014*, pages 197–212, 2014.
3. H. Betz, F. Gropengießer, K. Hose, and K. Sattler. Learning from the History of Distributed Query Processing – A Heretic View on Linked Data Management. In *COLD'12*, 2012.
4. C. Bondiombouy and P. Valduriez. Query processing in multistore systems: an overview. *IJCC*, 5(4):309–346, 2016.
5. A. Charalambidis, A. Troumpoukis, and S. Konstantopoulos. SemaGrow: Optimizing Federated SPARQL queries. In *SEMANTICS'15*, pages 121–128, 2015.
6. J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres, and M. Arias. Binary RDF representation for publication and exchange (HDT). *J. Web Sem.*, 19:22–41, 2013.
7. O. Görlitz and S. Staab. Federated Data Management and Query Optimization for Linked Open Data. In *New Directions in Web Data Management 1*, pages 109–137. 2011.
8. O. Görlitz and S. Staab. SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In *COLD'11*, 2011.
9. A. Gubichev and T. Neumann. Exploiting the query structure for efficient join ordering in SPARQL queries. In *EDBT'14*, pages 439–450, 2014.
10. S. Hagedorn, K. Hose, K. Sattler, and J. Umbrich. Resource Planning for SPARQL Query Execution on Data Sharing Platforms. In *COLD*, pages 49–60, 2014.
11. A. Harth, K. Hose, and R. Schenkel. Database Techniques for Linked Data Management. In *SIGMOD*, 2012.
12. A. Harth, K. Hose, and R. Schenkel. *Linked Data Management*. Chapman and Hall/CRC, 2014.
13. O. Hartig and C. B. Aranda. Bindings-Restricted Triple Pattern Fragments. In *OTM 2016 Conferences*, pages 762–779, 2016.
14. O. Hartig, C. Bizer, and J. C. Freytag. Executing SPARQL Queries over the Web of Linked Data. In *ISWC 2009*, pages 293–309, 2009.
15. O. Hartig, I. Letter, and J. Pérez. A Formal Framework for Comparing Linked Data Fragments. In *ISWC 2017*, pages 364–382, 2017.

16. J. V. Herwegen, R. Verborgh, E. Mannens, and R. V. de Walle. Query Execution Optimization for Clients of Triple Pattern Fragments. In *ESWC*, pages 302–318, 2015.
17. D. Ibragimov, K. Hose, T. B. Pedersen, and E. Zimányi. Processing Aggregate Queries in a Federation of SPARQL Endpoints. In *ESWC*, pages 269–285, 2015.
18. G. Montoya, H. Skaf-Molli, and K. Hose. The Odyssey Approach for Optimizing Federated SPARQL Queries. In *ISWC*, pages 471–489, 2017.
19. G. Montoya, H. Skaf-Molli, P. Molli, and M. Vidal. Decomposing federated queries in presence of replicated fragments. *J. Web Sem.*, 42:1–18, 2017.
20. T. Neumann and G. Moerkotte. Characteristic Sets: Accurate Cardinality Estimation for RDF Queries with Multiple Joins. In *ICDE'11*, pages 984–994, 2011.
21. J. Pérez, M. Arenas, and C. Gutiérrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, 2009.
22. M. Saleem and A. N. Ngomo. HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation. In *ESWC*, pages 176–191, 2014.
23. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization Techniques for Federated Query Processing on Linked Data. In *ISWC'11*, pages 601–616, 2011.
24. R. Taelman, J. V. Herwegen, M. V. Sande, and R. Verborgh. Comunica: a Modular SPARQL Query Engine for the Web. In *ISWC*, 2018. To appear.
25. J. Umbrich, K. Hose, M. Karnstedt, A. Harth, and A. Polleres. Comparing data summaries for processing live queries over Linked Data. *World Wide Web*, 14(5-6):495–544, 2011.
26. R. Verborgh, M. V. Sande, O. Hartig, J. V. Herwegen, L. D. Vocht, B. D. Meester, G. Haesendonck, and P. Colpaert. Triple Pattern Fragments: A low-cost knowledge graph interface for the Web. *J. Web Sem.*, 37-38:184–206, 2016.