Towards Efficient Query Processing over Heterogeneous RDF Interfaces

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

Aalborg University, Denmark
{gmontoya@cs,caebel13@student,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 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 answer sizes varying from 1 up to 1 million, show execution time gains of up to three orders of magnitude and data transfer reduction of up to four orders of magnitude.

1 Introduction

Efficiently evaluating SPARQL queries over heterogeneous RDF interfaces, such as SPARQL endpoints and Triple Pattern Fragments (TPFs) [19], requires considering their individual characteristics through all the 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 [14]. A well-known heuristic for executing joins is to reduce the sizes intermediate results, in particular when intermediate results have to be transferred between clients and servers [7]. As in [7], this paper focuses on BGP queries to ease the presentation of the proposed approach and provide insightful empirical results. We believe that our approach and the gained insights are also applicable to more complex fragments of the SPARQL query language.

The goal of this paper is to 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 a smart client that is aware of alternative interfaces for the same dataset, and that is able to use them to evaluate different parts of queries and combine partial answers (to subqueries) to produce query answers. In summary, this paper makes the following contributions:

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

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 presents our hybrid approach, Section 6 discusses our experimental results, and finally Section 7 concludes with an outlook to future work.

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 data structures (RDBMS, NoSQL, HDFS, etc.) depending on what would be the most efficient structure 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 [8], Triple Pattern Fragments [19], or bindings-restricted Triple Pattern Fragments [7]. These interfaces, in combination with different fragments of the SPARQL language, represent restrictions on expressiveness. Like with multistores, these interfaces 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. [9] 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 the 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 this interface can decrease the query response time and deteriorate the satisfaction of the users. To that end, using a model 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, our approach still relies on the same data model (RDF) but focuses on stores with different and in many cases complementary strengths. We focus on BGP queries (triple patterns with joins) to ease the presentation of the proposed approach and provide insightful empirical results. We believe that such an approach and the gained insights are applicable to more complex fragments of the SPARQL query language and will consider this in our future work.

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 [2, 5, 11, 12, 15, 16, 18, 20] and LDF [19]. 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 [11] or pruning sources that only provide redundant data [12]. In contrast to these techniques, the work presented in this paper consider heterogeneous interfaces that provide access to the same dataset. However, some of the query optimization strategies proposed in those works, could be combined with our approach. For instance, in combination with the optimizations proposed in [12], 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 [6].

3.2 Triple Pattern Fragment (TPF)

Verborgh et al. [19] 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 has been considerably reduced by using TPF interfaces, the paradigm used by TPF leads to less efficient query executions [7]. 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 [7] 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 [7].

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 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. Finally, if the number of clients is not so high and the server can invest some computational resources into processing queries that are a bit more complex in order to reduce the number of received requests, then the brTPF interface is the best option.

Table 1: RDF Interfaces Overview

	Server side	Transferred data	Client side
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

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. Query processing time at server and client is determined by the data structures used to store intermediate results and during query processing, e.g., available indexes, the sizes of the intermediate results, and the complexity of the algorithms used to process the query. Transfer time is mainly determined by the sizes of the (intermediate) results.

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 size of the query result, and query processing time at the client is very low (and usually negligible).

If the SPARQL endpoint had enough resources to instantly process every query of all users, then the best possible approach would be to use the SPARQL endpoint in all cases. However, the resources of SPARQL endpoints are limited and as such should be used in cases when using other interfaces would incur in higher response times.

Knowing the sizes of intermediate and query results could help deciding which interface to use to execute the queries. But having this knowledge before execution time is usually unrealistic. Therefore, we rely on heuristics based on metadata already available during execution time and that does not entail any additional computational costs. The most basic of such data is the number of triples in a given triple pattern fragment, a generally good enough estimation of this value is provided by the TPF server whenever a triple pattern fragment is retrieved. If the estimation is low (lower than a given threshold), 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 estimation 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.

To illustrate the impact of the interfaces on the query efficiency and resource usage, we will use queries generated using the WatDiv benchmark [3] 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 fragment sizes 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 (see Section 6 for more details about the setup).

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 (Listing1.1) has small fragments (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 cardinalities estimations that are deteriorated by the different types of joins and the unsatisfied assumption of query triple pattern independence [13].

Query B (Listing1.2) involves only fragments with relatively large sizes and many subject-subject joins on the same variables. Then, using brTPF results in relatively high amounts of data transfer in comparison to 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 more clients issuing queries, the servers represent bottlenecks, which can result in higher response times.

5 Efficient Query Execution over Available Interfaces

The goal of a heuristic approach is to use a combination of multiple interfaces to minimize 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

Table 2: Fragment sizes, number of results (NR), and evaluation metrics execution time (ER) and number of transferred bytes (NB) for Queries A and B

(a) Fragment sizes

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	NR	4 clients				16 clients			
		Endpoint		brTPF		Endpoint		brTPF	
		ET	NB	ET	NB	ET	NB	ET	NB
Query A	30,386	139,676	10,178,792	47,536	12,094,040	175,734	10,178,792	95,674	12,094,040
Query B	383	241	227,338	36,237	3,820,006	359	227,338	286,812	3,820,006

Algorithm 1 Evaluate BGP

Input: 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:     tps ← orderTriplePatterns(tps)
6:     if fragmentSize(first(tps)) > threshold ∧ cardinality(tps) > 1 then
7:         ms' ← evaluateAtSPARQLEndpoint(tps)
8:         ms ← ms' ⋈ ms
9:     else
10:        first ← first(tps)
11:        tps ← removeFirst(tps)
12:        ms' ← evaluateAtBrTPFServer(first, ms)
13:        ms ← ms' ⋈ ms
14:        evaluateBGP(tps, ms,threshold)
15:    end if
16:    return ms
17: end function
```

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 our approach implementing the above mentioned heuristics. Different ordering strategies can be used in Line 5 to decide the execution order of the triple patterns. The execution order used in this paper is as follows: The first triple pattern is the one with lowest fragment size. The second triple pattern is the one with the greater number of bound variables, when considering the variables given from the first triple pattern, and so on. As such, it is more likely that fragment sizes will be reduced as the number of bound variables increases. Note, that it is possible to use any other execution order for triple

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.
}
```

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 fragment size 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 size of the fragment corresponding to the most selective triple pattern 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 6). For Query A (Listing1.1) the first triple pattern (*tp1*) is evaluated first and the 47 values of *?v1* are used to evaluate *tp2* as a brTPF; the 47 values are used to build two brTPF requests with maximum 30 bindings per request. As the resulting fragments have sizes 26 and 15, the evaluation continues using the brTPF approach. The 41 values obtained for *?v2* are used to build two brTPF requests to evaluate *tp3*. As the retrieved fragments have sizes 489 and 113, thresholds of up to 112 results 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 TPF server.

Table 3: Number of results (NR), and evaluation metrics execution time (ER) and number of transferred bytes (NB) for Queries A and B using the hybrid approach

Query	4 clients				16 clients			
	ET	NB Endpoint	NB brTPF	NB	ET	NB Endpoint	NB brTPF	NB
Query A	20,191	10,665,981	21,103	10,687,084	15,362	10,665,981	21,103	10,687,084
Query B	333	233,478	0	233,478	547	233,478	0	233,478

Table 3 shows the evaluation metrics for the hybrid approach obtained when executing queries A and B (Listings 1.1 and 1.2) in the same setups as for the execution with brTPF and endpoint (metrics 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 the hybrid approach, 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.

6 Evaluation

To evidence the potential of our proposed approach, we have designed a prototype system 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 [3] query generator and a generated 10 million triples dataset to conduct an experiment with multiple clients – similarly as done in [7]. To show that our approach works well with a wide range of queries, we use queries with different answer sizes[1]; we consider queries with answer sizes in the ranges (1,100], (100, 1,000], (1,000, 10,000], (10,000, 100,000], and (100,000, 1,000,000]. We consider up to 32 clients, having each client 200-203 different queries to execute, i.e., in total 6,437 queries are executed in parallel in the setup with 32 clients.

Implementation: The hybrid engine is implemented in Node.js on top of the brTPF client[2], and is available at https://github.com/gmontoya/hybridSPARQLEngine.

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. *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 sum of the number of queries executed by all the clients in one minute.

[1] Queries are available at our github repository under queries/queriesWatDivEvaluation.tar.gz

[2] Available at http://olafhartig.de/brTPF-ODBASE2016/

6.1 Experimental Results

To evaluate the performance of our hybrid engine, 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. For the hybrid approach, 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 the hybrid engine 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.[3] 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 number of transferred bytes (NTB). For instance, in Figure 3, instead of showing around 3,200 points per approach (one for each query), we depict only 18 points, where each point represents the average value for a group of queries with average NTB in the interval [x*10^6,y*10^6) for the label [x,y) in the x-axis of Figure 3. Detailed results are available at our github repository.

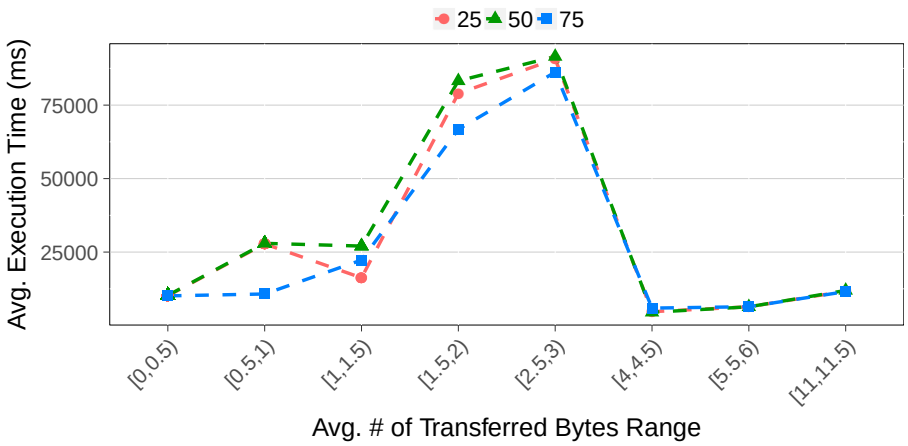


Fig. 1: Average Execution Time in ms (ET), 16 clients setup (x-axis, *10^6)

Impact of the threshold on the hybrid approach performance Figure 1 shows the average execution time obtained when using different values of threshold in the setup with 16 clients. We observe that as the threshold is increased,

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

the performance of our approach improves in average. Though, 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.

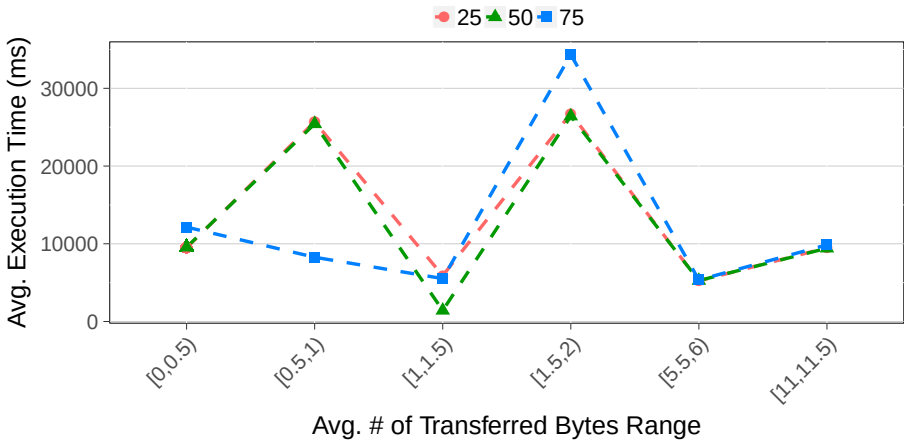


Fig. 2: Average Execution Time in ms (ET), 4 clients setup(x-axis, $*10^6$)

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

Execution time Figure 3 shows the average execution time obtained for the hybrid approach and the brTPF and endpoint baselines 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, the hybrid approach has exploited the advantages of each of these approaches to rely on brTPF only for very simple subqueries with one triple pattern or access relatively small triple pattern fragments that contribute to reduce 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, for the hybrid approach.

Number of Transferred Bytes Figure 4 shows the average number of transferred bytes for the hybrid approach and the brTPF and endpoint baselines in the setup with 16 clients. Clearly the endpoint baseline transfers the lowest

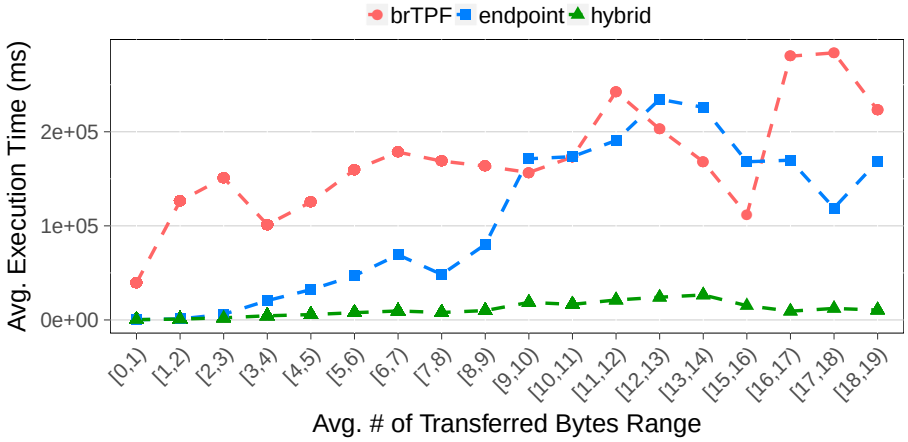


Fig. 3: Average Execution Time in ms (ET), 16 clients setup for BrTPF, Endpoint, and our hybrid approach (x-axis, $*10^6$)

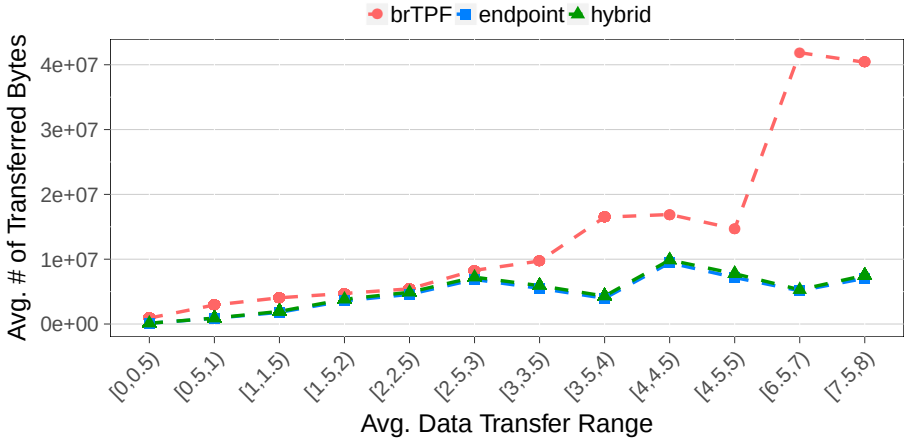
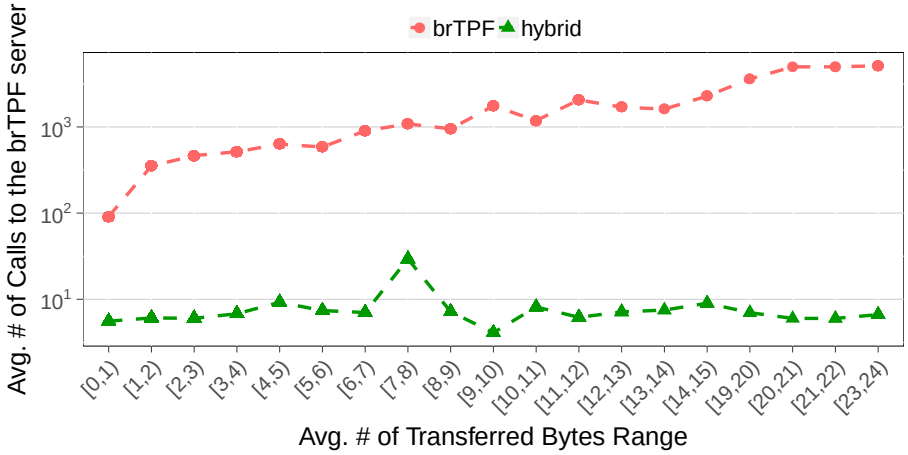
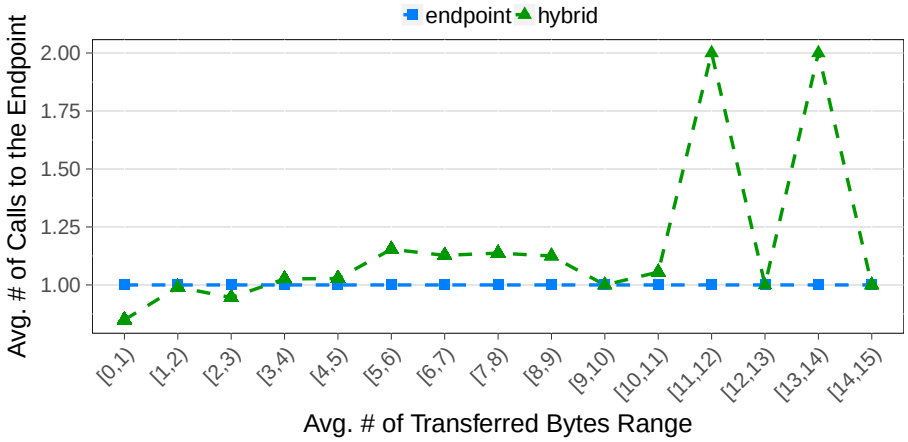


Fig. 4: Total Number of Transferred Bytes from the TPF server and the SPARQL Endpoint (NTB), 16 clients setup for BrTPF, Endpoint and hybrid approaches (x-axis, $*10^4$)

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



(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, $*10^7$) and to the SPARQL Endpoint (NCSE, a-xis, $*10^6$), 16 clients setup for BrTPF, Endpoint and hybrid approaches

Number of Calls Figure 5 shows the average number of calls to the TPF server and to the SPARQL endpoint for the hybrid approach and the brTPF and endpoint baselines in the setup with 16 clients. Even if the hybrid approach heavily relies on the SPARQL endpoint that can provide better performance for executing the queries, for the queries with fragment sizes 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.

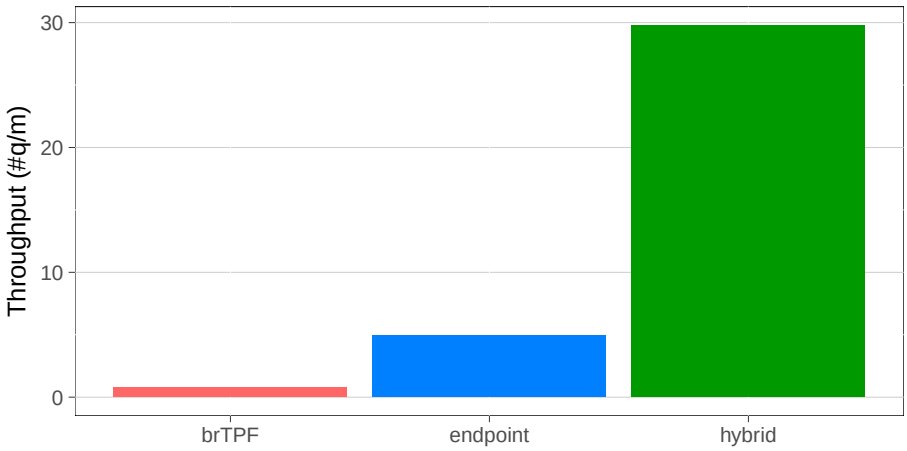


Fig. 6: System throughput, 16 clients setup for BrTPF, Endpoint and hybrid approaches

System Throughput Figure 6 shows system throughput for the hybrid approach and the brTPF and endpoint baselines in the setup with 16 clients. The throughput of the hybrid approach is considerably higher than the other approaches. The hybrid approach 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 of making better use of these resources.

7 Conclusion and Future Work

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

As future work, we plan to extend our approach to more complex fragments of the SPARQL query language. Additionally, we would like to combine our approach with other interfaces, such as other TPF clients, e.g., [1, 10]. Finally, we

would like to integrate our techniques into the Comunica platform [17]. Comunica allows to integrate different querying 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 our hybrid approach into Comunica, will facilitate the use of diverse RDF interfaces that access the same datasets and the use of the latest implementations of the query engines our approach relies upon.

Acknowledgment

This research was partially funded by the Danish Council for Independent Research (DFF) under grant agreement no. DFF-4093-00301.

References

1. M. Acosta and M. Vidal. Networks of linked data eddies: An adaptive web query processing engine for RDF data. In *ISWC*, pages 111–127, 2015.
2. 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.
3. G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee. Diversified Stress Testing of RDF Data Management Systems. In *ISWC 2014, Part I*, pages 197–212, 2014.
4. C. Bondiombouy and P. Valduriez. Query processing in multistore systems: an overview. *IJCC*, 5(4):309–346, 2016.
5. 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.
6. A. Gubichev and T. Neumann. Exploiting the query structure for efficient join ordering in SPARQL queries. In *EDBT'14*, pages 439–450, 2014.
7. O. Hartig and C. B. Aranda. Bindings-restricted triple pattern fragments. In *On the Move to Meaningful Internet Systems: OTM 2016 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2016, Rhodes, Greece, October 24-28, 2016, Proceedings*, pages 762–779, 2016.
8. O. Hartig, C. Bizer, and J. C. Freytag. Executing SPARQL queries over the web of linked data. In *The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings*, pages 293–309, 2009.
9. O. Hartig, I. Letter, and J. Pérez. A formal framework for comparing linked data fragments. In *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*, pages 364–382, 2017.
10. 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.
11. G. Montoya, H. Skaf-Molli, and K. Hose. The odyssey approach for optimizing federated SPARQL queries. In *ISWC*, pages 471–489, 2017.
12. 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.

13. T. Neumann and G. Moerkotte. Characteristic Sets: Accurate Cardinality Estimation for RDF Queries with Multiple Joins. In *ICDE'11*, pages 984–994, 2011.
14. J. Pérez, M. Arenas, and C. Gutiérrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, 2009.
15. M. Saleem and A. N. Ngomo. HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation. In *ESWC*, pages 176–191, 2014.
16. 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.
17. R. Taelman, J. V. Herwegen, M. V. Sande, and R. Verborgh. Comunica: a Modular SPARQL Query Engine for theWeb. In *ISWC*, 2018. To appear.
18. 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.
19. 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.
20. X. Wang, T. Tiropanis, and H. C. Davis. LHD: Optimising Linked Data Query Processing Using Parallelisation. In *LDOW*, 2013.