Towards More Intelligent SPARQL Querying Interfaces

Hashim Khan

Data Science Group, Paderborn University, Germany
hashim.khan@uni-paderborn.de

Abstract. Over years, the Web of Data has grown significantly. Various interfaces such as SPARQL endpoints, data dumps, and Triple Pattern Fragments (TPF) have been proposed to provide access to this data. Studies show that many of the SPARQL endpoints have availability issues. The data dumps do not provide live querying capabilities. The TPF solution aims to provide a trade-off between the availability and performance by dividing the workload among TPF servers and clients. In this solution, the TPF server only performs the triple patterns execution of the given SPARQL query. While the TPF client performs the joins between the triple patterns to compute the final resultset of the SPARQL query. High availability is achieved in TPF but increase in network bandwidth and query execution time lower the performance. We want to propose a more intelligent SPARQL querying server to keep the high availability along with high query execution performance, while minimizing the network bandwidth. The proposed server will offer query execution services (can be single triple patterns or even join execution) according to the current status of the workload. If a server is free, it should be able to execute the complete SPARQL query. Thus, the server will offer execution services while avoiding going beyond the maximum query processing limit, i.e. the point after which the performance start decreasing or even service shutdown. Furthermore, we want to develop a more intelligent client, which keeps track of a server's processing capabilities and therefore avoid DOS attacks and crashes.

Keywords: Availability, Performance, Intelligent TPF Server

1 Problem Statement

The problem we want to address focuses on the trade-off between the availability and performance in Linked Data interfaces.

A large amount of Linked Data is available on the web and it keeps on increasing day by day. According to LODStats[1], a total of $\sim$ 150 billion triples available from 9960 datasets. Querying this massive amount of data in a scalable way is particularly challenging. SPARQL is the primary query language to retrieve data from RDF linked datasets [28]. The true value of these datasets

[1] LODStats: `http://lodstats.aksw.org/`

becomes apparent when users can execute arbitrary queries over them, to retrieve precisely those facts they are interested in [28]. Various interfaces have been proposed to provide access to this data:

SPARQL endpoints offer a public interface to execute SPARQL queries over the underlying RDF data hosted by endpoints. In this interface, the client sends a complete SPARQL query to the server (i.e., SPARQL endpoint). The server executes the complete query and returns the final results. The responsibility of query execution is completely on the server side, while the client is idle most of the time. Here, better performance in query execution is ensured due to the use of indexes for query plan generation. Also, the network overhead is reduced as compared to the TFP server because in a case of SPARQL endpoint, only the final result of the query need to be transferred over the network. However many of the current SPARQL endpoints suffer from low availability rates due to its expressiveness towards query execution of all types [28,8]. A recent study [26] shows that more than 60% of endpoints are offline.

Triple Pattern Fragments interface tries to tackle the problem of SPARQL endpoints and aims to provide a trade-off between the availability and performance. The TPF interface divides the workload among servers and clients. In this solution, unlike SPARQL endpoints, the TPF server does not execute the complete SPARQL query. Rather, the server only performs the triple patterns execution of the given SPARQL query. While the TPF client performs the joins between the triple patterns to compute the final resultset of the SPARQL query. Study [28] shows the TPF interface achieves high availability at the expense of increased bandwidth and slower query execution times. This is because the client fetches the results of the complete triple patterns of the query and performs local joins. As such, it is possible that most of the triple pattern results are excluded after performing joins with the results of another triple pattern. Furthermore, in the case of TPF, the server selects and fetches the triples according to the triple patterns given in the query. After that, the server transfers these triples to the client for further execution, that is, to apply joins on it and to generate the final results. The approach is inefficient because the server's only job is to fetch the triples but not to execute the actual query. Also, transferring a large number of triples towards the client, makes the network overloaded which is an obvious reason for its low performance.

Data dumps offer another option to provide access to Linked Data. In this interface, the data is made publicly available as data dumps. The data dumps first need to be downloaded and later be processed on local machines to execute SPARQL queries. But this way of accessing data goes against the real aim of Semantic Web, i.e. it is not live querying.

Our Proposal: The main objective of this research is to provide a more intelligent TPF interface in which the server and client negotiate first, and then divide the task of query processing, according to the current status of the server.

Thus, it is not hard coded that TPF server will only execute triple patterns and TPF client will perform the joins between triple patterns. In the proposed interface, the server can even execute the complete SPARQL query, provided that current workload is not reaching beyond its processing capabilities. In this proposed work, we aim to achieve high availability as well as good query runtime performance.

2 Preliminaries

In this section, we formally define key definitions and basic concepts, which are used in this research proposal and my future PhD work .

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

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

Definition 3 (Solution Mapping). *A mapping μ from V to RT is a partial function $\mu : V \rightarrow RT$ where $RT = (I \cup B \cup L)$. For a triple pattern pattern obtained by replacing the variables in tp according to μ. The domain of μ, denoted by $dom(\mu)$, is the subset of V where μ is defined. We sometimes write down concrete mappings in square brackets, for instance, $\mu = [?X \rightarrow a, ?Y \rightarrow b]$ is the mapping with $dom(\mu) = \{?X, ?Y\}$ such that, $\mu(?X) = a$ and $\mu(?Y) = b$.*

Definition 4 (Triple Pattern Matching). *Let d be a dataset (we call it data source as well) with set of RDF as the set of mappings $[\![tp]\!]d = \{\mu : V \rightarrow RT \,|\, dom(\mu) = var(P) \, and \, \mu(P) \subseteq d\}$. If $\mu \in [\![tp]\!]d$, we say that μ is a solution for tp in d. If a data source d has at least one solution for a triple pattern tp, then we say d matches tp.*

Definition 5 (Linked Data Fragment). *According to [28], "a Linked Data Fragment (LDF) of a dataset is a resource consisting of those triples of this dataset that match a specific selector, together with their metadata and hypermedia controls to retrieve other Linked Data Fragments. We define a specific type of ldfs that require minimal effort to generate by a server, while still enabling efficient querying on the client side".*

Definition 6 (Triple Pattern Fragment). *According to [28], "a Triple Pattern Fragment (TPF) is a Linked Data Fragment with a triple pattern as selector, count metadata, and the controls to retrieve any other triple pattern fragment of the dataset. Each page of a triple pattern fragment contains a subset of the matching triples, together with all metadata and controls".*

Now we explain the query execution process in TPF interfaces. For basic graph patterns (BGPs), the algorithm works as follows:

1. For each triple pattern tp_i in the BGP B = $\{tp_1, \dots, tpn\}$, fetch the first page $\phi_1^i$of the triple pattern fragment f_i for tp_i, which contains an estimate cnt_i of the total number of matches for tp_i. Choose $\in$ such that $cnt_\in = min(cnt_1, \dots, cnt_n)$. $f_\in$ is then the optimal fragment to start with.
2. Fetch all remaining pages of the triple pattern fragment $f_\in$. For each triple $t \in LDF$, generate the solution mapping μ_t such that $\mu_t(tp_\in) =$ t. Compose the subpattern Bt $= \{tp|tp = \mu_t(tp_j) \wedge tp_j \in B\} \setminus \{t\}$. If Bt $\neq \emptyset$, find mappings Ω_{B_t} by recursively calling the algorithm for B_t. Else, $\Omega_{B_t} = \{\mu_\emptyset\}$ with $\mu_\emptyset$ the empty mapping.
3. Return all solution mappings $\mu \in \{\mu_t \cup \mu' | \mu' \in \Omega_{B_t}\}$.

2.1 Use Case Scenario

Consider the following SPARQL query to be executed on TPF interface of the DBpedia TPF [2], to find countries and their capitals of Europe:

```
PREFIX dbo:<http://dbpedia.org/ontology/>
PREFIX dct:<http://purl.org/dc/terms/>
PREFIX dbr:<http://dbpedia.org/resource/>

SELECT ?countries ?city
WHERE {
    ?countries dbo:capital ?city . #tp1 = 111
    ?countries dct:subject dbr:Category:Countries_in_Europe. #tp2 = 1741
}
```

In case of TPF server it only performs the triple patterns execution, that is, it has returned 111 triples which matched with the triple pattern 1 (tp1) while that of 1741 with triple pattern 2 (tp2). After the server fetches these 111 + 1741 = 1851 triples in total, the client performs the join between the triple patterns to compute the final resultset of the SPARQL query, which has only 48 results [3]. Fetching so many triples to generate such a small result, clearly shows that the high availability benefits of the TPF come at the expense of increased bandwidth and slower query execution times.

The proposed TPF server will offer query execution services (can be single triple patterns or even join execution) according to the current status of the workload. If a server is free, it should be able to execute the complete SPARQL query. Thus, the server will offer execution services while avoiding reaching the maximum query processing limit, i.e. service shutdown. This type of service is possible with coordination between server and client in an intelligent way, which

[2] DBpedia TPF: `http://fragments.dbpedia.org/`
[3] As a result of this query, we got 48 number of result rows.

is the aim of this research. In the motivating example, if the server is able to execute the complete query, then only the final query result (i.e., 48 in total) will be fetched across the network.

3 Relevancy

Performance can be defined as the rate at which a task can be completed. In our case, performance is the number of querying executed by the proposed interface in a time interval. Availability is the characteristic of a server where it can listen to client request and respond well in time. Due to the increasing trend of the Semantic Web, developers of the applications care about a reliable source which can ensure the provision of linked data in a live queryable form.

High performance with high availability is the requirement of almost every data consumer. Whenever there is a large amount of data, people will want to query it - and nothing is more intriguing than the vast amounts of Linked Data published over the last few years [5]. Application developers require Linked Data which should be available and can be queried in an efficient way.

As discussed in section 1, existing interfaces offer either good performance or availability but not both. In this research our aim is to make a trade-off between performance and availability, which is shown in Fig. 1.

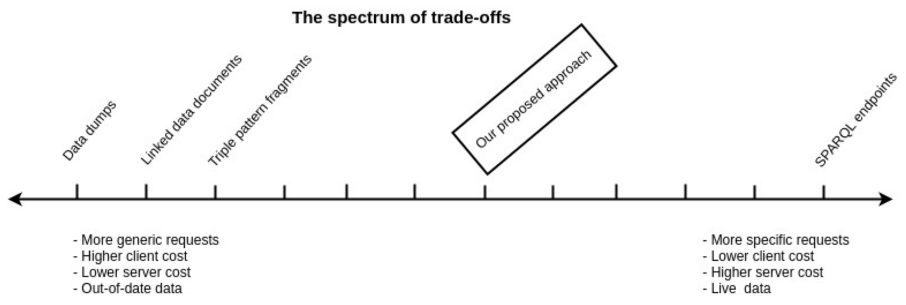

Fig. 1. An abstract placement of the proposed interface on the trade-off spectrum line

4 Related Work

The easiest way to publish the Linked Data knowledge graph is to upload an archive, usually called Data dumps, in an RDF format such as N-Triples or N-Quads. Clients download these files through HTTP and use it according to their needs. Advantage of using Data dump is its universality, i.e. client obtains whole of the knowledge graph and then makes it available online through any access point of its choice locally [29]. The issue here is that this is not live querying. Some initiatives have been taken to ease the usage of data dumps. For example, the LOD Laundromat [4] obtains data dumps from the Web, cleans up quality issues like serialization, and then republishes the dumps in the same RDF format. However, this effort still does not handle the issue that data is not live queryable: clients are still bound to download data dumps and upload them in a SPARQL-based system before SPARQL queries can be executed.

SPARQL endpoint uses the SPARQL protocol, which follows the First-Come-First-Served principle, therefore the queries have to wait in queue for execution by the server [4].Many triple stores, like Virtuoso [10] and Jena TDB [1], offer a SPARQL interface.While serving queries concurrently by a SPARQL endpoint, it has to restrict the size of the result-set returned to the end user or to generate a time-out error message in a case where a query spends too many resources [8]. Therefore, we may not get the complete result-set which we should expect from the query. Also, a characteristic of SPARQL endpoints is that, in addition to the uncertainty of the total number of requests to be made by different users, the execution cost per request varies significantly due to the relatively high expressiveness of SPARQL [20]. In 2013 a survey disclosed that most of the public SPARQL endpoints had an uptime of less than 95% [27].

In order to tackle these issues, most public LOD providers introduced a policy for the fair utilisation of the endpoint resources. As an example, DBpedia administrator shared the specifications relating to public SPARQL endpoint utilization. According to this specification, a query can have a maximum of 120 seconds execution time, 10000 results and 50 parallel connections per IP address[5]. Since every data provider configures the endpoint of their own, thus it will be difficult for application developers to rely on it.

Another strategy in this regard is to decompose a query into sub-queries and then to execute it under quotas [3]. The main shortcomings of this approach are (i) knowing the quotas configured by the data provider is not always possible and (ii) these quotas can not be applied to all the SPARQL queries to get the correct evaluation result.

Many other approaches exist to apply queries over Linked Data documents [13]. One type of approach makes use of pre-populated index structures [25] and another focuses on live exploration by the use of traversal-based query execution [15]. Typically, these query execution approaches have longer query execution times compared to SPARQL endpoints, but—unlike data dumps—permit for live querying. The required bandwidth is generally less than that of the data dumps, but the efficiency can still be low depending on the type of query. Moreover, completeness with respect to a knowledge graph cannot be ensured, and some of the queries are difficult or impossible to evaluate without the use of an index [13]. As an example, queries for patterns with unbound subjects (e.g., ?s foaf:made <o>) show problems because Linked Data documents, by its definition are subject-centric.

[4] `https://www.w3.org/TR/sparql11-overview/`

[5] `https://wiki.dbpedia.org/public-sparql-endpoint`

The Triple Pattern Fragments (TPF) approach [29] is an implementation of the Linked Data Fragments (LDF) [16], where the server's only job is to evaluate triple pattern queries. In this approach, the triple pattern queries can be executed in bounded time [17], therefore the TPF server does not face a convoy effect [7]. However, as joins have to be performed on the client side, a huge amount of triples transfer from the server to the client leading to poor SPARQL query execution performance.

Hartig et al. further extended the idea of basic TPF in brTPF [14], where by the use of well-known bind join strategy [12], it allows clients to attach intermediate results to TPF requests. The response to such a brTPF request is expected to contain RDF triples from the underlying dataset, that do not only match the given triple pattern (as in the case of TPF), but also guarantees the contribution in join with the given intermediate result. Hence, given the brTPF interface, it becomes possible to distribute the execution of joins between client and server. Due to this approach the overall throughput of the client-server system is significantly reduced.

Minier et al. proposed a SPARQL query engine based on Web preemption [18]. It allows SPARQL queries to be suspended by the web server after a fixed time quantum and resumed upon client request. This approach has solved the problem of partial result generation but the server may not perform well, when all the parallel queries are of high cost at a time.

The related work discussed here shows that, for the application developers to rely on linked open data, there exists a gap between the live availability and the efficient query execution. Therefore, in this the author aims to fill this gap.

5 Research Questions

In order to balance the trade-off between *availability* and *performance*, through intelligent TPF server and client, we have to answer the following research questions:

I- How can we define a *Workload Threshold (WT)* beyond which the server query execution performance starts decreasing?

II- How to avoid the WT?

III- How the server should communicate with the client to convey its current status? What services it can provide to the client?

IV- How the client will break the query into blocks according the the current capabilities of the server?

6 Hypothesis

Our hypotheses are directly related to the performance and high availability:

H1. Performance: The *null* hypothesis pertaining to query runtime performance states that the proposed approach would not lead to significant performance improvements with respect to the state-of-the-art triple pattern fragment interfaces. The *alternative* hypothesis states that the proposed approach would lead to a significant performance improvement with respect to the state-of-the-art triple pattern fragment interfaces.

H2. Availability: The *null* hypothesis pertaining to server availability states that the proposed approach would not lead to availability comparable to the state-of-the-art triple pattern fragment interfaces. The *alternative* hypothesis states that the proposed approach would be able to achieve availability, comparable to the state-of-the-art triple pattern fragment interfaces.

7 Approach

The approach proposed in this research work aims to provide a trade-off between availability and performance by dividing the workload among the servers and clients. Therefore, it is important to discuss the responsibilities of the client and server.

7.1 Server's Responsibility

The main functions of the proposed server are:

1. **Workload Monitoring:** As discussed, the proposed server offers services according to the current workload status. Therefore, the server should be able to monitor its current status of the workload and avoid reaching the maximum WT limit.
2. **Workload-aware Services:** The server provides query execution services according to the current status of the workload. As such, it is possible that the server will execute the complete SPARQL query (like SPARQL endpoints) or only execute individual triple patterns (like TPF), depending on the current status of the workload. Thus, it is also possible that the query execution plan is divided among the server and client, i.e., some of the query joins are performed by the server while others are executed by the client.
3. **Coordination:** As the proposed server provides adaptive query execution services, it is important to first negotiate a connection with client and coordinate the current execution services that are available for the given connection. The client then needs to decompose the given input query according to the available services.

7.2 Client's responsibility

The main functions of the client are:

1. **Coordination:** As discussed previously, the client has to take updates of server's current available services and decompose the given SPARQL query accordingly.
2. **Query Planning** The query execution plan will be generated on the client's side according to the current available services from the server. The partial query execution can also be done by the client.

At present, we are working on devising algorithms to perform the above mentioned responsibilities. The first step is to know the WT for the different available interfaces. To this end, we are conducting experiments by using Iguana [9], a benchmark execution framework to analyze the performance of triplestores using multiple parallel agents with/without updates. The goal of these experiments is to calculate the maximum WT limit. We are increasing workload by using multiple querying agents and analyzing the overall throughput until it reaches the maximum limit. Once, we know the maximum WT limit for the proposed server, the next step would be to make the server more intelligent, able to avoid reaching the maximum WT limit.

8 Evaluation Plan

In this section, we briefly explain our evaluation plan to compare the proposed approach with state-of-the-art approaches. Please note that this plan may changes according to the availability of new benchmarks, performance metrics, and new solutions for SPARQL query processing.

Benchmark Selection: The first important step in our evaluation is the selection of the most relevant and representative SPARQL benchmark. To this end, SPARQL benchmarks such as BSBM [6], LUBM [11], DBpedia SPARQL benchmarks [19], WatDiv [2], SP2Bench [23], and FEASIBLE [21] are generally used in state-of-the-art evaluations of SPARQL querying interfaces. Recent study [22] shows that synthetic benchmarks can be used to test the scalability of the systems. However, they often fail to contain important SPARQL query constructs such `UNION`, `FILTER`, `OPTIONAL` etc. Furthermore, the study reveals that the DBpedia benchmark generated by FEASIBLE [21] framework is the most diverse SPARQL benchmark on the specified SPARQL features. We plan to use the most state-of-the-art WatDiv (synthetic) and FEASIBLE (real data and queries) benchmarks in our evaluation.

Metrics: In general, a SPARQL querying benchmark comprises of three main components: (1) a set of RDF datasets, (2) a set of SPARQL queries, and (3) a set of performance metrics [22]. The first two components are directly provided by the selected benchmark. There can be many performance metrics as discussed in [22]. We will mostly focus on the metrics relevant to the availability and performance of the SPARQL querying interfaces.

The metrics relevant to the query runtime performances are: (1) query runtime, (2) Query Mix per Hour (QMpH), (3) Queries per Second (QpS), (4) Number of intermediate results and (5) Network traffic generated by the interfaces [21,19,6]. The metrics relevant to the availability of the systems are: (1) Query processing overhead in terms of the CPU and memory usage (this also includes the number of disk/memory swaps), (2) Parallelism with/without Updates which measure the parallel query processing capabilities of the querying interfaces by simulating workloads from multiple querying agents with and without dataset updates [9,30,6].

Systems: SAGE [18] and Communica [24] are state-of-the-art HDT-based interfaces which we will compare with the proposed solution. In addition, SPARQL endpoints interfaces backed by triplestores such as Virtuoso, BlazeGraph, GraphDB etc. will also be considered in the evaluation.

Best Practices: We will follow the standard best practices such as running benchmark queries multiple times and presenting their average results, starting with a warm up phase where certain test queries will be run before starting the actual evaluation etc. Furthermore, we will use various significance tests such as T-Test, Wilcoxon signed-rank test etc. to measure the significance of the performance improvements. To ensure the reproducibility of our results, we will make sure to provide public access to the complete data, queries, and the results used in our evaluation. Furthermore, we will create a user manual to help in reproducing the results.

9 Reflections

We believe that the real essence of the Semantic Web will be felt when semantically Linked Data is available, and applications can easily and reliably access the very specific part of that data, in an efficient manner. Our proposed approach takes into account providing not only availability of Linked Data or efficient execution of the queries, but providing both at the same time, by balancing trade-offs between the two aspects.

10 Acknowledgments

This work has been supported by the German Federal Ministry of Transport and Digital Infrastructure (BMVI) in the projects LIMBO (no. 19F2029I) and OPAL (no. 19F2028A), the H2020 Marie Skłodowska-Curie project KnowGraphs (GA no. 860801), and the German Federal Ministry of Education and Research (BMBF) within the program 'KMU-innovativ: Forschung für die zivile Sicherheit' in the project SOLIDE (no. 13N14456).

I am thankful to my PhD supervisors Prof. Dr. Axel-cyrille Ngonga Ngomo and Dr. Muhammad Saleem.

References

1. SIGUCCS '09: Proceedings of the 37th Annual ACM SIGUCCS Fall Conference: Communication and Collaboration. ACM, New York, NY, USA (2009), 459091
2. Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified stress testing of RDF data management systems. In: ISWC, pp. 197–212 (2014). https://doi.org/10.1007/978-3-319-11964-9_13, https://doi.org/10.1007/978-3-319-11964-9_13
3. Aranda, C.B., Polleres, A., Umbrich, J.: Strategies for executing federated queries in sparql1.1. In: International Semantic Web Conference (2014)
4. Beek, W., Rietveld, L., Bazoobandi, H.R., Wielemaker, J., Schlobach, S.: Lod laundromat: A uniform way of publishing other people's dirty data. In: Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., Goble, C. (eds.) The Semantic Web – ISWC 2014. pp. 213–228. Springer International Publishing, Cham (2014)
5. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. Int. J. Semantic Web Inf. Syst. **5**(3), 1–22 (2009)
6. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. Int. J. Semantic Web Inf. Syst. **5**(2), 1–24 (2009). https://doi.org/10.4018/jswis.2009040101, https://doi.org/10.4018/jswis.2009040101
7. Blasgen, M., Gray, J., Mitoma, M., Price, T.: The convoy phenomenon. SIGOPS Oper. Syst. Rev. **13**(2), 20–25 (Apr 1979). https://doi.org/10.1145/850657.850659, http://doi.acm.org/10.1145/850657.850659
8. Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.Y.: Sparql web-querying infrastructure: Ready for action? In: International Semantic Web Conference. pp. 277–293. Springer (2013)
9. Conrads, F., Lehmann, J., Saleem, M., Morsey, M., Ngomo, A.N.: IGUANA: A generic framework for benchmarking the read-write performance of triple stores. In: ISWC. pp. 48–65. Springer (2017). https://doi.org/10.1007/978-3-319-68204-4_5, https://doi.org/10.1007/978-3-319-68204-4_5
10. Erling, O., Mikhailov, I.: Virtuoso: RDF Support in a Native RDBMS, pp. 501–519. Springer Berlin Heidelberg, Berlin, Heidelberg (2010), https://doi.org/10.1007/978-3-642-04329-1_21
11. Guo, Y., Pan, Z., Heflin, J.: LUBM: a benchmark for OWL knowledge base systems. J. Web Sem. **3**(2-3), 158–182 (2005). https://doi.org/10.1016/j.websem.2005.06.005, https://doi.org/10.1016/j.websem.2005.06.005
12. Haas, L.M., Kossmann, D., Wimmers, E.L., Yang, J.: Optimizing queries across diverse data sources. In: Proceedings of the 23rd International Conference on Very Large Data Bases. pp. 276–285. VLDB '97, San Francisco, CA, USA (1997), http://dl.acm.org/citation.cfm?id=645923.670995
13. Hartig, O.: An overview on execution strategies for linked data queries. Datenbank-Spektrum **13**(2), 89–99 (Jul 2013). https://doi.org/10.1007/s13222-013-0122-1, https://doi.org/10.1007/s13222-013-0122-1
14. Hartig, O., Aranda, C.B.: brtpf: Bindings-restricted triple pattern fragments (extended preprint). CoRR (2016), http://arxiv.org/abs/1608.08148
15. Hartig, O., Bizer, C., Freytag, J.C.: Executing sparql queries over the web of linked data. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) The Semantic Web - ISWC 2009. pp. 293–309. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

16. Hartig, O., Letter, I., Pérez, J.: A formal framework for comparing linked data fragments. In: d'Amato, C., Fernandez, M., Tamma, V., Lecue, F., Cudré-Mauroux, P., Sequeda, J., Lange, C., Heflin, J. (eds.) The Semantic Web – ISWC 2017. pp. 364–382. Springer International Publishing, Cham (2017)
17. Heling, L., Acosta, M., Maleshkova, M., Sure-Vetter, Y.: Querying large knowledge graphs over triple pattern fragments: An empirical study. In: Vrandečić, D., Bontcheva, K., Suárez-Figueroa, M.C., Presutti, V., Celino, I., Sabou, M., Kaffee, L.A., Simperl, E. (eds.) The Semantic Web – ISWC 2018. pp. 86–102. Springer International Publishing, Cham (2018)
18. Minier, T., Skaf-Molli, H., Molli, P.: Sage: Web preemption for public sparql query services. In: The World Wide Web Conference. pp. 1268–1278. WWW '19, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3308558.3313652, http://doi.acm.org/10.1145/3308558.3313652
19. Morsey, M., Lehmann, J., Auer, S., Ngomo, A.N.: DBpedia SPARQL benchmark - performance assessment with real queries on real data. In: ISWC. pp. 454–469. Springer (2011). https://doi.org/10.1007/978-3-642-25073-6_29, https://doi.org/10.1007/978-3-642-25073-6_29
20. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of sparql. ACM Trans. Database Syst. **34**(3), 16:1–16:45 (Sep 2009). https://doi.org/10.1145/1567274.1567278
21. Saleem, M., Mehmood, Q., Ngomo, A.N.: FEASIBLE: a feature-based SPARQL benchmark generation framework. In: ISWC. pp. 52–69. Springer (2015)
22. Saleem, M., Szárnyas, G., Conrads, F., Bukhari, S.A.C., Mehmood, Q., Ngonga Ngomo, A.C.: How representative is a sparql benchmark? an analysis of rdf triplestore benchmarks. In: The World Wide Web Conference. pp. 1623–1633. WWW '19, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3308558.3313556, http://doi.acm.org/10.1145/3308558.3313556
23. Schmidt, M., et al.: SP2Bench: A SPARQL performance benchmark. In: Semantic Web Information Management - A Model-Based Perspective, pp. 371–393 (2009), https://doi.org/10.1007/978-3-642-04329-1_16
24. Taelman, R., Van Herwegen, J., Vander Sande, M., Verborgh, R.: Comunica: A modular sparql query engine for the web. In: Vrandečić, D., Bontcheva, K., Suárez-Figueroa, M.C., Presutti, V., Celino, I., Sabou, M., Kaffee, L.A., Simperl, E. (eds.) The Semantic Web – ISWC 2018. pp. 239–255. Springer International Publishing, Cham (2018)
25. 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 (Oct 2011). https://doi.org/10.1007/s11280-010-0107-z, https://doi.org/10.1007/s11280-010-0107-z
26. Vandenbussche, P.Y., Umbrich, J., Matteis, L., Hogan, A., Buil-Aranda, C.: Sparqles: Monitoring public sparql endpoints. Semantic web **8**(6), 1049–1065 (2017)
27. Vandenbussche, P.Y., Umbrich, J., Matteis, L., Hogan, A., Buil-Aranda, C.: Sparqles: Monitoring public sparql endpoints. Semantic Web **8**, 1–17 (01 2017). https://doi.org/10.3233/SW-170254
28. Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., Sande, M.V., Cyganiak, R., Colpaert, P., Mannens, E., Van De Walle, R.: Low-cost queryable linked data through triple pattern fragments. In: Proceedings of the 2014 International Conference on Posters &#38; Demonstrations Track - Volume 1272. pp. 13–16. ISWC-PD'14, CEUR-WS.org, Aachen, Germany, Germany (2014), http://dl.acm.org/citation.cfm?id=2878453.2878457

29. Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., Colpaert, P.: Triple pattern fragments: a low-cost knowledge graph interface for the web. JOURNAL OF WEB SEMANTICS **37-38**, 184–206 (2016), `http://dx.doi.org/10.1016/j.websem.2016.03.003`
30. Wu, H., et al.: BioBenchmark Toyama 2012: An evaluation of the performance of triple stores on biological data. J. Biomedical Semantics **5**, 32 (2014). https://doi.org/10.1186/2041-1480-5-32, `https://doi.org/10.1186/2041-1480-5-32`