SPARQL Endpoint Metrics for Quality-Aware Linked Data Consumption

Johannes Lorey
Hasso Plattner Institute
Potsdam, Germany
johannes.lorey@hpi.uni-potsdam.de

ABSTRACT

In recent years, dozens of publicly accessible Linked Data repositories containing vast amounts of knowledge presented in the Resource Description Framework (RDF) format have been set up worldwide. By utilizing the SPARQL query language, users can consume, integrate, and present data from a federation of sources for different application scenarios. However, several challenges arise for distributed query processing across multiple SPARQL endpoints, such as devising suitable query optimization or result caching strategies.

For implementing these techniques, one crucial aspect lies in determining appropriate endpoint features. In this work, we introduce several metrics that enable universal and fine-grained characterization of arbitrary Linked Data repositories. We present comprehensive approaches for deriving these metrics and validate them through extensive evaluation on real-world SPARQL endpoints. Finally, we discuss possible implications of our findings for data consumers.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*

General Terms

Performance, Measurement, Design

Keywords

Distributed query processing, Linked Data, RDF, Semantic Web, SPARQL, Quality of Service metrics

1. INTRODUCTION

The SPARQL Protocol and RDF Query Language (SPARQL) is considered one of the core technologies of the Semantic Web. Retrieving Linked Data through public SPARQL endpoints represents a novel form of information dissemination: Traditionally, accessing certain databases by issuing structured queries (e.g., SQL queries) has been restricted to a limited number of users, e.g., within a company network. Public SPARQL endpoints on the other hand enable for a large number of users to access information from multiple sources simultaneously. Related Semantic Web technologies, such as well-defined ontologies, simplify the integration process in comparison to classical data integration efforts that rely on undocumented or cryptic schema information.

However, as several popular publicly available Linked Data repositories have been set up as proof-of-concept in the context of research projects (e.g., DBpedia[1], LinkedGeoData[2], LinkedMDB[3], . . .), retrieving information from those SPARQL endpoints at large scale is cumbersome. Typically, different policies are implemented to limit the number of requests or the amount of retrievable information. For example, to prevent malicious attacks and ensure responsiveness, the popular DBpedia SPARQL endpoint is configured to return at most 50,000 result rows or 10 MB per request while allowing only a maximum number of 15 requests per second [8]. Additionally, as oftentimes SPARQL endpoints are deployed on commodity hardware using off-the-shelf frameworks, they are typically not configured to process specific workloads as efficiently as possible, e.g., when compared to databases maintained in an enterprise context.

Thus, when querying and integrating Linked Data from a federation of SPARQL endpoints, one challenge lies in determining appropriate endpoint characteristics, e.g., for devising suitable query plans. Here, derived statistics need to accurately reflect actual SPARQL operations instead of accounting only for generic access patterns when querying the endpoint, thus potentially underestimating execution costs. For example, while the general network latency for connecting to an endpoint might be low, the delay for sending SPARQL requests and retrieving appropriate responses might be considerably higher. This can be caused by other processes running on the endpoint or misconfigured set-ups. Also, even though executing a simple request can result in reasonable response time, more complex queries containing joins may take considerably longer to execute, e.g., due to missing indexes or unexpected query engine behavior.

In this work, we propose a novel idea for determining SPARQL endpoint characteristics that aid data consumption in general and data integration in particular. To this end, we present a number of comprehensive metrics suitable for describing more complex workloads of SPARQL queries. Here, we focus on several operations that have high relevance for real-world requests as indicated by previous findings [2].

[1] `http://dbpedia.org/sparql`
[2] `http://linkedgeodata.org/sparql`
[3] `http://data.linkedmdb.org/sparql`

Moreover, we present approaches to determine these metrics using suitable SPARQL queries. Thus, we argue that the results determined by issuing these queries accurately reflect actual SPARQL requests. We back up this argument by evaluating the introduced metrics using real-world SPARQL endpoints and provide a thorough discussion of the results.

This paper is organized as follows: We present related work for our approach in Sec. 2, where we also point out novel aspects of our metrics. Next, we introduce the proposed metrics and their implementation in Sec. 3. Afterwards, we evaluate these metrics on several real-world SPARQL endpoints and discuss the findings in Sec. 4. Finally, in Sec. 5 we summarize this paper and indicate future research potential.

2. RELATED WORK

The related work for this paper draws mainly from two fields: First, in recent years several frameworks for distributed SPARQL processing have been devised. Here, we comment on similarities and differences between these projects and our approach. Also, multiple benchmarks have been established for evaluating the performance of RDF storage systems. We illustrate key characteristics of these benchmarks and point out the connection to this paper.

2.1 Distributed SPARQL Processing

Whereas conventional relational database management systems (RDBMS) typically rely on a shared nothing architecture mostly for replication and load-balancing purposes, Linked Data repositories allow for truly distributed query processing. *DARQ* [12], one early project assisting distributed SPARQL querying, provides a transparent layer that enables access to a federation of SPARQL endpoints. Here, the goal is to minimize the overall query execution time by determining the optimal execution plan. The information used for this task is based on statistics included in the service description for each endpoint. *SPLENDID* [5], a similar framework, uses voiD [1] descriptions instead. However, in real-world scenarios, this kind of metainformation is oftentimes not available, insufficient, or outdated.

A more recent project named *FedX* [15] allows for efficient distributed SPARQL query processing without the need of any explicit service annotations or voiD descriptions. Instead, extensive operator re-ordering techniques are applied: For example, a rule-based optimizer is utilized for ordering joins according to heuristics determined in advance. Whereas this approach does not rely on metadata published by the endpoint provider, it also does not factor in endpoint characteristics at run-time which might be useful for ad-hoc fine-tuning the execution strategy.

The common goal of these projects is to optimize federated SPARQL query processing, e.g., by determining the most suitable query plan or modifying the query structure. The corresponding approaches rely on different information, such as provided service descriptions (*DARQ*, *SPLENDID*) or pre-computed metadata (*FedX*). Conversely, our work aims at determining generic characteristics of real-world SPARQL endpoints that allow to predict the performance of individual operations (such as joins). On the one hand, the goal of our work is to assist data consumers in the evaluation of the quality of service of publicly available SPARQL endpoints. On the other hand, the performance metrics introduced in this paper can also be considered as additional input features for other distributed query processing frameworks.

2.2 RDF Benchmarking

The Berlin SPARQL Benchmark (*BSBM*) [4] is one of the earliest frameworks for comparing the performance of different RDF datastores. In this benchmark, various systems are analyzed using synthetic workloads mimicking typical operations in an e-commerce scenario. A similar project entitled *SP^2Bench* [14] utilizes DBLP publication records for generating workloads instead. Whereas the majority of queries in *BSBM* contain fairly simple SPARQL expressions, *SP^2Bench* exploits the variety of complex SPARQL operators, such as `FILTER` expressions. However, as both benchmarks rely on very specific (synthetic) query workloads, their general eligibility for assessing query execution performance on real-world SPARQL endpoints is limited.

On the other hand, a number of alternative benchmarking frameworks aim at capturing actual Linked Data interactions. In [10], the authors examine normalized user queries issued against the popular DBpedia endpoint. Similar approaches targeting the generation of representative benchmark queries for concrete RDF knowledge bases are presented in [6, 13]. The goal of these works is to establish comprehensive workloads suitable for determining more realistic performance results. However, whereas the benchmarks presented in the previous paragraph are typically deemed too generic, approaches custom-tailored for specific real-world datasets (e.g., DBpedia) lack ubiquitous applicability.

In our work, we instead aim at providing a universal means for discerning characteristics of different SPARQL endpoints without the need for real-world query workloads. Here, we do not rank different fine-tuned datastores hosted within an isolated environment, but rather strive to capture the behavior of publicly available SPARQL endpoints without any a priori information about their configuration. As those endpoint typically employ some limitations on the amount of resources provided to parse individual requests, complex benchmark queries such as the ones presented in [14] potentially incur timeouts. Thus, we focus on representative queries adhering to SPARQL standards and argue that the aggregated individual results can serve as heuristics for estimating the performance of more intricate query workloads.

3. METRICS

In this section, we introduce the different metrics we consider as criteria for characterizing SPARQL endpoints. We point out the intuition of these metrics, their relationship to generic workloads, and comment on previous findings where applicable. Additionally, we illustrate how we determine the values for these metrics using actual SPARQL queries.

3.1 Latency

In our context, the latency of an endpoint is equal to the sum of the delay between the client sending a request and the SPARQL endpoint receiving it, and the delay between the endpoint sending the corresponding results and the client receiving them. Conversely, the latency excludes any time spent on query processing, e.g., for creating a query plan and executing it. Instead we aim at discovering the minimum time required for any issued valid SPARQL request. We use this information to normalize the values of all other metrics.

We are not only interested in determining the network latency itself (i.e., the round-trip time of the communication channel), but also in the delay of accessing the triple store via the SPARQL interface. Consequently, to measure the overall latency we employ queries that incur no or only negligible execution cost. In Query 1, we use the `ASK` query form that returns a Boolean result (which is `true` for any non-empty SPARQL endpoint). As `ASK` has been part of SPARQL since the first working draft[4] of the query language, we expect all endpoints to support corresponding queries.

```
ASK {
  ?s ?p ?o .
}
```

Query 1: Latency Query

The evaluation of any SPARQL query with only one triple pattern has complexity $\mathcal{O}(n)$ (cf. Theorem 1 in [7]), where n is the size of the dataset. Given that Query 1 always checks the first triple stored in the endpoint, evaluating this query is done in constant time. As the actual query result, i.e., the data sent back to the client, is of small size, this query should give a good indication of how high the latency is.

Latency in packet-switched networks is influenced by several factors, one of them being the (physical) distance of the communication channel between the sender and the receiver of the packets. In most cases, a packet will be forwarded over a number of intermediary links before reaching the receiver, thus the overall (minimum) latency is determined by the aggregated individual latencies between all hops. Whereas the well-known `ping` command can give a good indication of the general network latency on the network layer, this method underestimates the actual round-trip time for sending requests on the application layer (e.g., when issuing SPARQL queries).

3.2 Throughput

Measuring the throughput of a SPARQL endpoint indicates how much data can be transmitted over a certain period of time. In our case, the amount of data is represented by the number of bindings generated by the endpoint for the variables contained in the SPARQL query. We normalize the time measured for executing Query 2 by the actual number of result bindings received. Assuming a reasonably large dataset containing at least 1,000 triples, the total amount of (not necessarily unique) bindings for Query 2 should be 3,000 (1,000 for each of the variables `?s`, `?p`, `?o`).

```
SELECT * WHERE {
  ?s ?p ?o .
} LIMIT 1000
```

Query 2: Throughput Query

As with latency, throughput is influenced by the network infrastructure. In case either the receiver or any of the intermediary links experiences a high request load, throughput may suffer. Due to the best-effort characteristics of the Internet, typically no guarantees can be given for the achieved throughput between any two nodes in the network. However, as the number of potential routes for short-distance packet switching is smaller due to the lower number of intermediary hops, throughput can generally be estimated more reliably than for long-distance communication [11].

[4] `http://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/`

3.3 Execution Time of Joins

Actual SPARQL queries can be quite complex, e.g., with regard to the number of contained graph patterns. Thus, we base the execution time measurements on three elementary join graph patterns derived from the observations illustrated in [2]: The subject-subject-join, the object-object-join, and the subject-object-join. These graph pattern can give hints about certain endpoint characteristics, such as available indexes or selectivity of subjects and objects.

To determine the execution time of the join operations, we need to retrieve a number of sample triples on which they can be applied. As a reference, for the subject-subject-join operation, the appropriate sampling request is displayed in Query 3. Using the `FILTER` condition, we ensure that the retrieved bindings differ in at least the object, thus eliminating joins of identical triples. Typically, predicates are less selective than either subjects or objects in RDF statements [16], therefore in general non-equality can be identified more easily for objects than for predicates when issuing Query 3.

```
SELECT ?p1 ?p2 ?o1 ?o2 WHERE {
  ?s ?p1 ?o1 .
  ?s ?p2 ?o2 .
  FILTER (?o1 != ?o2)
}
```

Query 3: Subject-Subject-Join Sampling Query

We also randomize the requests that are used to retrieve sample data regarding the position of the first match. In general, we can use random values for the `OFFSET` operator to retrieve arbitrary information from the knowledge base, assuming the `OFFSET` position is smaller than the overall number of results for the query. In the case of Virtuoso-backed SPARQL endpoints, we also utilize the custom `bif:rnd` function which retrieves elements considerably faster than using high `OFFSET` values.

We devised Query 4, Query 5, and Query 6 to probe the execution time of the subject-subject-join, object-object-join, and subject-object-join, respectively. Here, resources in the queries are instantiated using the data retrieved during sampling. For instance, in Query 4 the resources p_1, o_1, p_2, o_2 are replaced by the corresponding results for Query 3.

```
SELECT ?s WHERE {
  ?s p1 o1 .
  ?s p2 o2 .
} LIMIT 1
```

Query 4: Subject-Subject-Join Query

```
SELECT ?o WHERE {
  s1 p1 ?o .
  s2 p2 ?o .
} LIMIT 1
```

Query 5: Object-Object-Join Query

```
SELECT ?so WHERE {
  ?so p2 o    .
  s   p1 ?so  .
} LIMIT 1
```

Query 6: Subject-Object-Join Query

4. EVALUATION

For our evaluation, we gathered results for the metrics described in Sec. 3 for three SPARQL endpoints: DBpedia, LinkedGeoData (LGD), and LinkedMDB. All of these have been established in the context of research projects. At the time of writing, the DBpedia and LinkedGeoData endpoints utilize the OpenLink Virtuoso framework[5] (Version 7.00 and 6.02, respectively), while LinkedMDB uses the D2R server [3] to allow data access via SPARQL.

First, we measured the metrics described in Sec. 3 by issuing the corresponding queries from a local machine running Microsoft Windows Server 2008 R2 and connected to the Internet through a 1 Gbps network interface. In addition, we conducted more measurements on the Elastic Compute Cloud[6] (EC2) provided by Amazon Web Services (AWS) to compare different locations and hardware resources. To this end, we instantiated the default Linux Amazon Machine Image[7] (AMI) in various configurations: "EU (Ireland) Tiny", "US (West) Tiny", and "US (West) Medium"[8]. Here, the "Tiny" and "Medium" instances differ mostly in the computing and network performance of their virtualized hardware.

For each set-up, we recorded 100 measurements for the metrics described in Sec. 3. For all our experiments, we randomized the order of the requests sent to the endpoint to reduce potential (short-term) server-side caching of results. Additionally, all experiments were run during a 24 hour interval to cater for potential access spikes at certain times in different parts around the globe. Consequently, the average delay between any two successive requests was approx. 22 seconds. Any request that did not yield an HTTP 200 response, e.g., because of transmission errors, was not included in the analyses. All sample bindings required for the join operations described in Sec. 3.3 were retrieved several days before conducting the actual experiments, thus eliminating any caching effects.

In Fig. 1-3, we visualize measurements for the different join operations w.r.t. the individual latency across all experimental set-ups for the DBpedia, LinkedGeoData, and LinkedMDB endpoints, respectively. For all measured values and endpoints, each column represents the upper quartile (Q_3) execution time for the respective operation and the average latency (grey), thus summarizing the round-trip time of the SPARQL queries introduced in Sec. 3.3. We illustrate the Q_3 values as they provide robust upper bound estimations while eliminating high-value outliers. We also indicate the average value of the individual execution times as a solid black line within the respective column.

The results in Fig. 1 exhibit great variation: Whereas the subject-subject-join execution times for the first three experiments (Local, EU Tiny, US Tiny) is similar, for the other two join operations the results differ significantly. However, it should be noted that the Q_3 execution time for the subject-subject-join operation was always higher than for the other two metrics. For the last set-up (US Medium), the Q_3 execution times were noticeably higher, possibly caused by high load experienced by the DBpedia endpoint at the time. Even though the results depicted in Fig. 1 are mixed, a general trend can be observed for these experiments: In

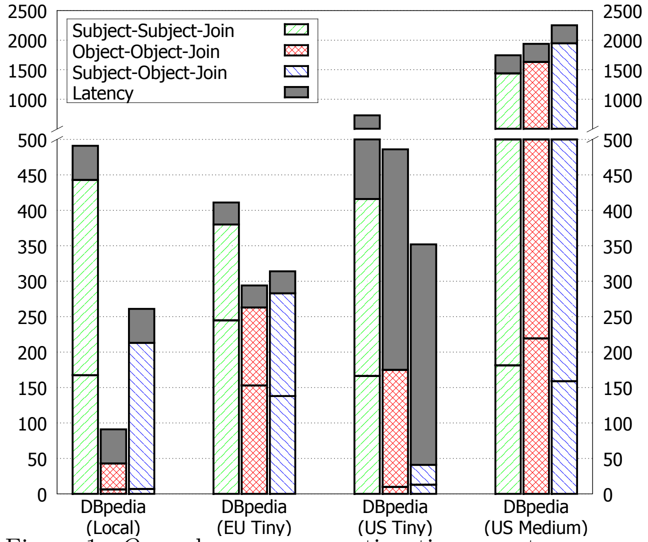

Figure 1: Q_3 and average execution times w.r.t. average latency for the DBpedia SPARQL endpoint (values in ms)

nearly all cases, the aggregated execution time, i.e., the sum of the join execution times and the latency, is higher than for any other endpoint. When considering the limitations placed on the DBpedia endpoint as outlined in [8], this observation suggests to replicate data locally when deploying time-critical Linked Data applications relying on this knowledge base. For instance, this can be done by caching retrieved results or by exploiting the provided file dumps.

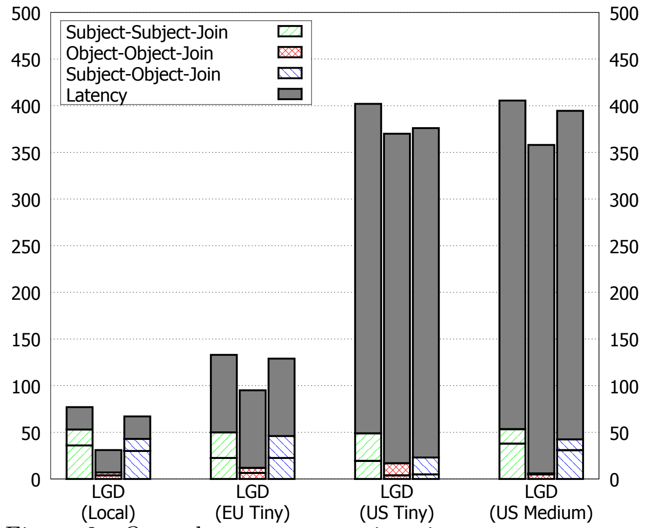

Figure 2: Q_3 and average execution times w.r.t. average latency for the LGD SPARQL endpoint (values in ms)

For LGD, the determined Q_3 and average values for all client configuration are remarkably similar as indicated in Fig. 2. Additionally, the general ratio between the different join operations remains nearly constant for all measurements. On the other hand, Fig. 2 illustrates the effect of varying latency when evaluating overall execution time: Whereas the different Q_3 execution time values are nearly identical among all set-ups, the latency of the EC2 US instances is in orders of magnitude higher than for the local machine or the EC2 instance hosted in the EU. In a real-world application this fact can assist in establishing suitable

[5] `http://virtuoso.openlinksw.com/`
[6] `http://aws.amazon.com/ec2/`
[7] `http://aws.amazon.com/amazon-linux-ami/`
[8] `http://aws.amazon.com/ec2/instance-types/`

caching strategies: For low-latency connections to the LGD endpoint, caching any data may not be necessary. However, retaining data locally might be beneficial when a client accessing the LGD endpoint incurs high latency.

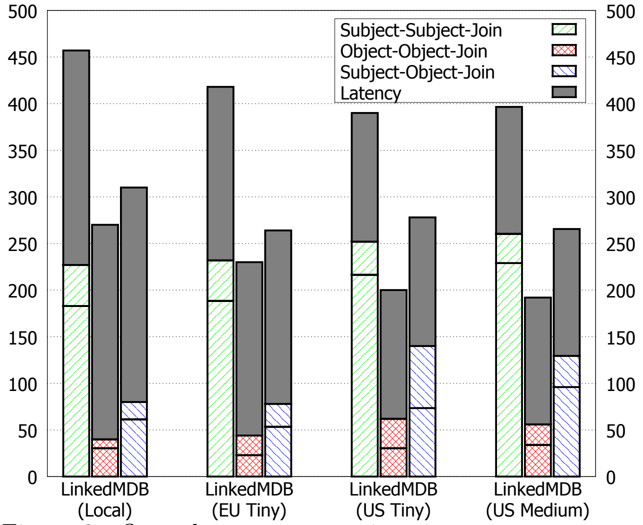

Figure 3: Q_3 and average execution times w.r.t. average latency for the LinkedMDB SPARQL endpoint (values in ms)

As with LGD, the ratio of the Q_3 execution time values between the different join operations for the LinkedMDB endpoint is nearly identical throughout our measurements as depicted in Fig. 3. Additionally, in our experiments the latency for connecting to this endpoint also remained nearly constant regardless of the client location. Thus, the overall execution times including latency for LinkedMDB are highly similar. Again, when consuming data from the LinkedMDB endpoint this insight might prove useful: If a client issues complicated requests (e.g., multiple subject-subject-joins), thereby aggregating costly execution time, it can help to store the received data locally for efficient future access.

5. CONCLUSION

In this work, we have presented several metrics aimed at characterizing SPARQL endpoints and evaluated these metrics for a number of publicly available Linked Data repositories. We determined in our evaluation that endpoints exhibit different characteristics: For instance, while it comes as no surprise that latency is influenced by the network infrastructure, the costs for join operations depend on a number of factors that are not obvious to a data consumer. However, by applying the metrics outlined in this work, he or she can determine whether an endpoint is eligible for certain types of services, e.g., for retrieving, processing, and presenting data in an interactive on-line application.

We illustrated several basic heuristics, which we consider essential buildings blocks for estimating the execution times of more complex workloads. Based on previous findings (e.g., as reported in [2]), the different operations underlying our metrics account for a large majority of all SPARQL queries. Consequently, the derived results are relevant for many application scenarios, e.g., for devising appropriate data caching, prefetching [9], and integration strategies, or generating query execution plans for federated systems.

For future work, we aim at incorporating more intricate request patterns as endpoint metrics while warranting universal applicability of our approach, e.g., with regard to the SPARQL 1.1 specification. Moreover, the conducted experiments can be extended to other endpoints to survey potential patterns between different SPARQL frameworks and relate these results to the findings established in RDF benchmarking experiments. Finally, we plan on deriving composition methods suitable for combining the proposed metrics to determine the cost of real-world SPARQL workloads.

6. REFERENCES

[1] K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. Describing linked datasets - on the design and usage of voiD, the "vocabulary of interlinked datasets". In *Proceedings of the WWW Workshop on Linked Data on the Web (LDOW)*, Madrid, Spain, 2009.

[2] M. Arias, J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente. An empirical study of real-world SPARQL queries. In *Proceedings of the International Workshop on Usage Analysis and the Web of Data*, Hyderabad, India, 2011.

[3] C. Bizer and R. Cyganiak. D2R server–publishing relational databases on the semantic web. In *Proceedings of the International Semantic Web Conference (ISWC)*, Athens, GA, USA, 2006.

[4] C. Bizer and A. Schultz. The Berlin SPARQL benchmark. *International Journal on Semantic Web and Information Systems*, 5(2):1–24, 2009.

[5] O. Görlitz and S. Staab. SPLENDID: SPARQL endpoint federation exploiting VOID descriptions. In *Proceedings of the International Workshop on Consuming Linked Data (COLD)*, Bonn, Germany, 2011.

[6] O. Görlitz, M. Thimm, and S. Staab. SPLODGE: Systematic generation of SPARQL benchmark queries for linked open data. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 116–132. Boston, MA, USA, 2012.

[7] C. G. Jorge Pérez, Marcelo Arenas. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3):16:1–16:45, 2009.

[8] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 2013. Under review.

[9] J. Lorey and F. Naumann. Detecting SPARQL query templates for data prefetching. In *Proceedings of the Extended Semantic Web Conference (ESWC)*, pages 124–139, Montpellier, France, 2013.

[10] M. Morsey, J. Lehmann, S. Auer, and A.-C. N. Ngomo. DBpedia SPARQL benchmark - performance assessment with real queries on real data. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 454–469, 2011.

[11] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE Network*, 17(6):27–35, 2003.

[12] B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In *Proceedings of the Extended Semantic Web Conference (ESWC)*, pages 524–538, Tenerife, Canary Islands, 2008.

[13] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. FedBench: A benchmark suite for federated semantic data query processing. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 585–600, Koblenz, Germany, 2011.

[14] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. Sp^{2}bench: A SPARQL performance benchmark. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 222–233, Shanghai, China, 2009.

[15] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: A federation layer for distributed query processing on linked open data. In *Proceedings of the Extended Semantic Web Conference (ESWC)*, pages 481–486, Heraklion, Greece, 2011.

[16] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds. SPARQL basic graph pattern optimization using selectivity estimation. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 595–604, Beijing, China, 2008.