Undefined 1 (2009) 1–5
IOS Press

A Fine-Grained Evaluation of SPARQL Endpoint Federation Systems

Editor(s): Name Surname, University, Country
Solicited review(s): Name Surname, University, Country
Open review(s): Name Surname, University, Country

Muhammad Saleem [a,*], Yasar Khan [b], Ali Hasnain [b], Ivan Ermilov [a], Axel-Cyrille Ngonga Ngomo [a]
[a] *Universität Leipzig, IFI/AKSW, PO 100920, D-04009 Leipzig*
E-mail: {saleem,ngonga,iermilov}@informatik.uni-leipzig.de
[b] *Insight Center for Data Analytics, National University of Ireland, Galway*
E-mail: {yasar.khan,ali.hasnain}@insight-centre.org

Abstract. The Web of Data has grown enormously over the last years. Currently, it comprises a large compendium of interlinked and distributed datasets from multiple domains. The abundance of datasets has motivated considerable work for developing SPARQL query federation systems, the dedicated means to access data distributed over the Web of Data. However, the granularity of previous evaluations of such systems has not allowed deriving of insights concerning their behavior in different steps involved during federated query processing. In this work, we perform extensive experiments to compare state-of-the-art SPARQL endpoint federation systems using the comprehensive performance evaluation framework FedBench. In addition to considering the tradition query runtime as an evaluation criterion, we extend the scope of our performance evaluation by considering criteria, which have not been paid much attention to in previous studies. In particular, we consider the number of sources selected, the total number of SPARQL ASK requests used, the completeness of answers as well as the source selection time. Yet, we show that they have a significant impact on the overall query runtime of existing systems. Moreover, we extend FedBench to mirror a highly distributed data environment and assess the behavior of existing systems by using the same four criteria. As the result we provide a detailed analysis of the experimental outcomes that reveal novel insights for improving current and future SPARQL federation systems.

Keywords: SPARQL federation, Web of Data, RDF

1. Introduction

The transition from the Web of Documents to the Web of Data has resulted in a large compendium of interlinked datasets from diverse domains. Currently, the Linked Open Data (LOD) Cloud[1] contains over 60 billion triples available from more than 1000 different datasets with large datasets [29] being added frequently. Due to the decentralized and linked architecture of LOD, answering complex queries often requires accessing and combining information from multiple datasets. Processing such *federated queries* [8,25,32,28] in a virtually integrated fashion is becoming increasingly popular. Given the importance of federated query processing over the Web of Data, it is critical to provide fine-grained evaluations to assess the quality of state-of-the-art implementations of federated SPARQL query engines by comparing them against common criteria through an open benchmark. Such fine-grained evaluation results are valuable when positioning new federation systems against existing. In addition, these results help application developers when choosing appropriate systems for a given application as they allow them to select federation systems through a comparison of their performance against their criteria of interest. Moreover, such fine-grained results provide useful insights for sys-

*Corresponding author. E-mail: saleem@informatik.uni-leipzig.de
[1] `http://stats.lod2.eu/`

0000-0000/09/$00.00 © 2009 – IOS Press and the authors. All rights reserved

tem developers and empower them to improve current federation systems as well as to develop better systems.

Current evaluations [8,1,32,26,22,27,36] of SPARQL query federation systems compare some of the federation systems based on the central criterion of overall query runtime. While optimizing the query runtime of federation systems is the ultimate goal of the research performed on this topic, the granularity of current evaluations fails to provide results that allow understanding why the query runtimes of systems can differ drastically and are thus insufficient to detect the components of systems that need to be improved. For example, key metrics such as a smart source selection in terms of the *total number of data sources selected*, the *total number of SPARQL ASK requests used*, and the *source selection time* have a direct impact on the overall query performance but are not addressed in the existing evaluations. For example, the over-estimation of the total number of relevant data sources will generate extra network traffic, may result in increased query execution time. The SPARQL ASK queries returns boolean value indicating whether a query pattern matches or not. The greater the number of SPARQL ASK requests used for source selection, the higher the source selection time and therefore overall query execution time. Furthermore, as pointed out by [23], the current testbeds [7,31,30,24] for evaluating, comparing, and eventually improving SPARQL query federation systems still have some limitations. Especially, the *partitioning of data* as well as *the SPARQL clauses* used cannot be tailored sufficiently, although they are known to have a direct impact on the behaviour of SPARQL query federation systems.

The aim of this paper is to experimentally evaluate a large number of SPARQL 1.0 query federation systems within a more fine-granular setting in which we can measure the time required to complete different steps of the SPARQL query federation process. To achieve this goal, we conducted a public survey[2] and collected information regarding 14 existing federated system implementations, their key features, and supported SPARQL clauses. Eight of the systems which participated in this survey are publicly available. However, two out of the eight with public implementation do not make use of the SPARQL endpoints and were thus not considered further in this study.

In the next step and like in previous evaluations, we compared the remaining six systems [8,25,32,1,36,21] with respect to the traditional performance criterion that is the query execution time using the commonly used benchmark FedBench. In addition, we also compared these six systems with respect to their *answer completeness*, source selection approach in terms of the *total number of sources* they selected, the *total number of SPARQL ASK requests* they used and *source selection time*. For the sake of completeness, we also performed a comparative analysis (based on the survey outcome) of the key functionality of the 14 systems which participated in our survey. The most important outcomes of this survey are presented in Section 3.[3]

To provide a quantitative analysis of the effect of *data partitioning* on the systems at hand, we extended both FedBench [30] and SP^{2}Bench [31] by distributing the data upon which they rely. To this end, we used the slice generation tool[4] described in [28]. This tool allows creating any number of subsets of a given dataset (called *slices*) while controlling the number of slices, the amount of overlap between the slices as well as the size distribution of these slices. The resulting slices were distributed across various data sources (SPARQL endpoints) to simulate a highly federated environment. In our experiments, we made use of both FedBench [30] and SP^{2}Bench [31] queries to ensure that we *cover the majority of the SPARQL query types and clauses*.

Our main contributions are summarized as follow:

- We present the results of a public survey which allows us to provide a crisp overview of categories of SPARQL federation systems as well as provide their implementation details, features, and supported SPARQL clauses.
- We present (to the best of our knowledge) most comprehensive experimental evaluation of open-source SPARQL federations systems in terms of their source selection and overall query runtime using in two different evaluation setups.
- Along with the central evaluation criterion (i.e., the overall query runtime), we measure three further criteria, i.e., the total number of sources selected, the total number of SPARQL ASK requests used, and the source selection time. By these means, we obtain deeper insights into the behaviour of SPARQL federation systems.

[2] Survey: `http://goo.gl/iXvKVT`, Results: `http://goo.gl/CNW5UC`

[3] All survey responses can be found at `http://goo.gl/CNW5UC`.

[4] `https://code.google.com/p/fed-eval/wiki/SliceGenerator`

- We extend both FedBench and SP^{2}Bench to mirror highly distributed data environments and test SPARQL endpoint federation systems for their parallel processing capabilities.
- We provide a detailed discussion of experimental results and reveal novel insights for improving existing and future federation systems.
- Our survey results points to research opportunities in the area of federated semantic data processing.

The rest of this paper is structured as follows: In Section 2, we provide an overview of state-of-the-art SPARQL federated query processing approaches. Section 3 provides a detailed description of the design of and the responses to our public survey of the SPARQL query federation. Section 4 provides an introduction to SPARQL query federation and selected approaches for experimental evaluation. Section 6 describes our evaluation framework and experimental results, including key performance metrics, a description of the used benchmarks (FedBench, SP^{2}Bench, SlicedBench), and the data slice generator. Section 7 provides our further discussion of the results. Finally, Section 8 concludes our work and gives an overview of possible future extensions.

2. Related work

In this section, we provide a two-pronged overview of existing works. First, we give an overview of existing federated SPARQL query system evaluations. Here, we focus on the description of different surveys/evaluations of SPARQL query federation systems and argue for the need of a new fine-grained evaluation of federated SPARQL query engines. Thereafter, we give an overview of benchmarks for SPARQL query processing engines. In addition, we provide reasons for our benchmark selection in this evaluation.

2.1. Federation systems evaluations

Several SPARQL query federation surveys have been developed over the last years. Rakhmawati et. al. [26] present a survey of SPARQL endpoint federation systems in which the details of the query federation process along with a comparison of the query evaluation strategies used in these systems. Moreover, systems that support both SPARQL 1.0 and SPARQL 1.1 are explained. However, this survey do not provide any experimental evaluation of the discussed SPARQL query federation systems. In addition, the system implementation details resp. supported features are not discussed in much detail. We address these drawbacks in Tables 1 resp. 2. Hartig et. al. [11] provide a general overview of Linked Data federation. In particular, they introduce the specific challenges that need to be addressed and focus on possible strategies for executing Linked Data queries. However, this survey do not provide an experimental evaluation of the discussed SPARQL query federation systems. Umbrich et. al. [16] provide a detailed study of the recall and effectiveness of Link Traversal Querying for the Web of Data. Schwarte et. al. [33] present an experimental study of large-scale RDF federations on top of the Bio2RDF data sources using a particular federation system, i.e., FedX [32]. They focus on design decisions, technical aspects, and experiences made in setting up and optimizing the Bio2RDF federation. [6] identifies various drawbacks of federated Linked Data query processing. The authors propose that Linked Data as a service has the potential to solve some of the identified problems. [10] presents limitations in Linked Data federated query processing and implications of these limitations. Moreover, this paper presents a query optimization approach based on semi-joins and dynamic programming. [19] identifies various strategies while processing federated queries over Linked Data. [35] provides an experimental evaluation of the different data summaries used in live query processing over Linked Data. [23] provides a detail discussion of the limitations of the existing testbeds used for the evaluation of SPARQL query federation systems. Some other experimental evaluations [8,1,32,36,22,27,23] of SPARQL query federation systems compare some of the federation systems based on their overall query runtime. For example, Gorlitz et. al. [8] compare their approach with three other approaches ([32,25], AliBaba[5], and RDF-3X 0.3.4.22[8]. An extension of ANAPSID presented in [22] compares ANAPSID with FedX using 10 FedBench-additional complex queries. FedX [32] compares its performance with AliBaba and DARQ using a subset of the FedBench queries. LHD [36] compares its performance with FedX, SPLENDID using the Berlin SPARQL Benchmark (BSBM) [7].

[5] `SesameAliBaba:http://www.openrdf.org/alibaba.jsp`) using a subset of the queries from FedBench. Furthermore, they measure the effect of the information in VoiD[6] descriptions on the accuracy of their source selection. Acosta et. al. [1] compare their approach performance with Virtuoso SPARQL endpoints, ARQ 2.8.8. BSD-style21[7]

[8] `http://www.mpi-inf.mpg.de/neumann/rdf3x/`

All experimental evaluations above compare only a small number of SPARQL query federation systems using a subset of the queries available in current benchmarks with respect to a single performance criterion (query execution time). Consequently, they fail to provide deeper insights into the behavior of these systems in different steps (e.g., source selection) required during the query federation. In this work, we evaluate six open-source federated SPARQL query engines experimentally on two different evaluation frameworks. To the best of our knowledge, this is currently the most comprehensive evaluation of open-source SPARQL query federation systems. Furthermore, along with central performance criterion of query runtime, we compare these systems with respect to their source selection. Our results show (section 6) that the different steps of the source selection affect the overall query runtime considerably. Thus, the insights gained through our evaluation w.r.t. to these criteria provide valuable insights for optimizing SPARQL query federation.

2.2. Benchmarks

Benchmarks for comparing SPARQL query processing systems have a rich literature as well. These include Berlin SPARQL Benchmark (BSBM), SP^{2}Bench, FedBench, Lehigh University Benchmark (LUBM) and the DBpedia Sparql Benchmark (DBPSB). Both BSBM and SP^{2}Bench are mainly designed for the evaluation of triple stores that keep their data in a single large repository. *BSBM* [7] was developed for comparing the performance of native RDF stores with the performance of SPARQL-to-SQL re-writers. *SP^{2}Bench* [31] mirrors vital characteristics (such as power law distributions or Gaussian curves) of the data in the DBLP bibliographic database. This benchmark comprises both a data generator for creating arbitrarily large DBLP-like documents and a set of carefully designed benchmark queries. FedBench [30] is designed explicitly to simulate SPARQL query federation tasks on real-world datasets with queries resembling typical requests on these datasets. Furthermore, this benchmark also includes a dataset and queries from SP^{2}Bench. LUBM [9] is designed to facilitate the evaluation of Semantic Web repositories in a systematic way. It is based on a customizable and repeatable synthetic data. DBPSB [24] includes queries from the DBpedia query log and aims to reflect the behavior of triple stores when confronted with real queries aiming to access native RDF data.

FedBench is the only (to the best of our knowledge) benchmark that encompasses real-world datasets, commonly used queries and distributed data environment. Furthermore, it is commonly used in the evaluation of SPARQL query federation systems [32,8,22,28]. Therefore, we choose this benchmark as a main evaluation benchmark in this paper. We also decided on using SP^{2}Bench in parts of our experiments to ensure that our queries cover most of SPARQL. Note that neither FedBench nor SP^{2}Bench provide SPARQL 1.1 federated queries. Devising such a benchmark remains future work.

3. Federated engines public survey

In order to provide a comprehensive overview of existing SPARQL federation engines, we designed and conducted a survey of SPARQL query federation engines. In this section, we present the principles and ideas behind the design of the survey as well as its results and their analysis.

3.1. Survey Design

The aim of the survey was to compare the existing SPARQL query federation engines, regardless of their implementation or code availability. To reach this aim, we interviewed domain experts and designed a survey with three sections: system information, requirements, and supported SPARQL clauses.[9]

The system information section of the survey includes implementation details of the SPARQL federation engine such as:

- **URL of the paper, engine implementation:** Provides the URL of the related scientific publication or URL to the engine implementation binaries/-code.
- **Code availability:** Indicates the disclosure of the code to the public.
- **Implementation and licensing:** Defines the programming language and distribution license of the framework.
- **Type of source selection:** Defines the source selection strategy used by the underlying federation system.
- **Type of join(s) used for data integration:** Shows the type of *joins* used to integrate sub-queries results coming from different data sources.

[9]The survey can be found at `http://goo.gl/iXvKVT`.

- **Use of cache:** Shows the usage of cache for performance improvement.
- **Support for catalog/index update:** Indicates the support for automatic index/catalog update.

The questions from the *requirements* section assess SPARQL query federation engines for the key features/requirements that a developer would require from such engines. These include:

- **Result completeness:** Given a SPARQL 1.0 query, can the federation engine retrieve all solutions for the given query (100% recall) or is it possible that it misses some of the solutions (for example due to the source selection or using an out-of-date index)?.
- **Policy-based query planning:** Most federation approaches target open data and do not provide restrictions (according to different user access rights) on data access during query planning. As a result, a federation engine may select a data source for which the requester is not authorized, thus overestimating the data sources and increasing the overall query runtime. Does your system have the capability of taking into account the privacy information (e.g., different graph-level access rights for different users, etc.) during query planning?
- **Support for partial results retrieval:** In some cases the query results can be too large and result completeness (i.e., 100% recall) may not be desired, rather partial but fast and/or quality query results are acceptable. Does the federation engine provide such functionality where a user can specify a desired recall (less than 100%) as a threshold for fast result retrieval? It is worth noticing that this is different from limiting the results using SPARQL LIMIT clause as it restricts the number of results to some fixed value while in partial result retrieval the number of retrieved results are relative to the actual total number of results.
- **Support for no-blocking operator/adaptive query processing:** SPARQL endpoints are sometimes blocked or down or exhibit high latency. Does the federation engine support non-blocking joins (where results are returned based on the order in which the data arrives, not in the order in which data being requested)?
- **Support for provenance information:** Usually, SPARQL query federation systems integrate results from multiple SPARQL endpoints without any provenance information, such as how many results were contributed by a given SPARQL endpoint or which of the results are contributed by each of the endpoint. Does the federation engine provide such provenance information?
- **Query runtime estimation:** In some cases a query may have a longer runtime (e.g., in the order of minutes). Does the federation engine provide means to approximate and display (to the user) the overall runtime of the query execution in advance?
- **Duplicate detection:** Due to the decentralized architecture of Linked Data Cloud, a sub-query might retrieve results that were already retrieved by another sub-query. For some applications, the former sub-query can be skipped from submission (federation) as it will only produce overlapping triples. Does the federation engine provide such a duplicate-aware SPARQL query federation? Note that this is the duplicate detection before sub-query submission to the SPARQL endpoints and the aim is to minimize the number of sub-queries submitted by the federation engine.
- **Top-K query processing:** Is the federation engine able to rank results based on the user's preferences (e.g., his/her profile, his/her location, etc.)?

The *supported SPARQL clauses* section assess existing SPARQL query federation engines w.r.t. the list of supported SPARQL clauses. The list of the SPARQL clauses is mostly based on the characteristics of the BSBM benchmark queries [7]. The summary of the used SPARQL clauses can be found in Table 3.

The survey was open and free for all to participate in. To contact potential participants, we used Google Scholar to retrieve papers that contained the keywords ŚPARQLánd qúery federation. After a manual filtering of the results, we contacted the main authors of the papers and informed them of the existence of the survey while asking them to participate. Moreover, we sent messages to the W3C Linked Open Data mailing list[10] and Semantic Web mailing list[11] with a request to participate. The survey was opened for two weeks.

3.2. Discussion of the survey results

Based on our survey results[12], existing SPARQL query federation approaches can be divided into three main categories (see Table 1)

[10] public-lod@w3.org
[11] semantic-web@w3.org
[12] Available at http://goo.gl/CNW5UC

Table 1

Overview of implementation details of federated SPARQL query engines (**SEF** = SPARQL Endpoints Federation, **DHTF** = DHT Federation, **LDF** = Linked Data Federation, **C.A.** = Code Availability, A.G.P.L. Affero General Public License, L.G.P.L. = Lesser General Public License, **S.S.T.** = Source Selection Type, **I.U.** = Index/catalog Update, (A+I) = SPARQL ASK and Index/catalog, (C+L) = Catalog and online discovery via Link-traversal), VENL = Vectored Evaluation in Nested Loop, **AGJ** = Adaptive Group Join, **ADJ** = Adaptive Dependent Join, **RMHJ** = Rank-aware Modification of Hash Join, NA = Not Applicable

Systems	Category	C.A	Implementation	Licencing	S.S.T	Join Type	Cache	I.U
FedX [32]	SEF	✓	Java	GNU A.G.P.L	index-free	bind (VENL)	✓	NA
LHD [36]	SEF	✓	Java	MIT	hybrid (A+I)	hash/ bind	✗	✗
SPLENDID [8]	SEF	✓	Java	L.G.P.L	hybrid (A+I)	hash/ bind	✗	✗
FedSearch [4]	SEF	✗	Java	GNU A.G.P.L	hybrid (A+I)	bind (VENL), pull based rank join (RMHJ)	✓	NA
GRANATUM [12,14]	SEF	✗	Java	yet to decide	index only	nested loop	✗	✗
Avalanche [5]	SEF	✗	Python, C, C++	yet to decide	index only	distributed, merge	✓	✗
DAW [28]	SEF	✗	Java	GNU G.P.L	hybrid (A+I)	based on underlying system	✓	✗
ANAPSID [1]	SEF	✓	Python	GNU G.P.L	hybrid (A+I)	AGJ, ADJ	✗	✓
ADERIS [21]	SEF	✓	Java	Apache	Index only	index-based nested loop	✗	✗
DARQ [25]	SEF	✓	Java	GPL	Index only	nested loop, bound	✗	✗
LDQPS [19]	LDF	✗	Java	Scala	hybrid (C+L)	symmetric hash	✗	✗
SIHJoin [20]	LDF	✗	Java	Scala	hybrid (C+L)	symmetric hash	✗	✗
WoDQA [2]	LDF	✓	Java	GPL	hybrid (A+I)	nested loop, bound	✓	✓
Atlas [18]	DHTF	✓	Java	GNU L.G.P.L	Index only	SQLite	✗	✗

Query federation over multiple SPARQL endpoints: In this type of approaches, RDF data is made available via SPARQL endpoints. The federation engine makes use of endpoint URLs to federate sub-queries and collect results back for integration. The advantage of this category of approaches is that queries are answered based on original, up-to-date data with no synchronization of the copied data required [11]. Moreover, the execution of queries can be carried out efficiently because the approach relies on SPARQL endpoints. However, such approaches are unable to deal with the data provided by the whole of LOD Cloud because sometimes Linked Data is not exposed through SPARQL endpoints.

Query federation over Linked Data: This type of approaches relies on the Linked Data principles[13] for query execution. The set of data sources which can contribute results is determined by using URI lookups during the query execution itself. Query federation over Linked Data does not require the data providers to publish their data as SPARQL endpoints. Instead, the only requirement is that the RDF data follows the Linked Data principles. A downside of these approaches is that they are less time-efficient than the previous approaches due to the URI lookups they perform.

Query federation on top of Distributed Hash Tables: This type of federation approaches stores RDF data on top of Distributed Hash Tables (DHTs) and use DHT indexing to federate SPARQL queries over multiple RDF nodes. This is a space-efficient solution and can reduce the network cost as well. However, many of the LOD datasets are not stored on top of DHTs.

Each of the above main category can be further divided into three sub-categories (see Table 1):

Catalog/index-assisted solutions: These approaches utilize dataset summaries that have been collected in a pre-processing stage. These approaches may lead to more efficient query federation. However, the index needs to be constantly updated to ensure complete results retrieval. The index size should also be kept to a minimum to ensure that it does not significantly increase the overall query processing costs.

Catalog/index-free solutions: In these approaches, the query federation is performed without using any stored data summaries. The data source statistics can be collected on the fly before the query federation starts. This approach promises that the results retrieved by the engine are complete and up-to-date. However, it may increase the query execution time, depending on the

[13] http://www.w3.org/DesignIssues/LinkedData.html

extra processing required for collecting and processing on-the-fly statistics.

Hybrid solutions: In these approaches, some of the data source statistics are pre-stored while some are collected on the fly, e.g., using SPARQL ASK queries.

Table 1 provides a classification along with the implementation details of the 14 systems which participated in the survey. Overall, we received responses mainly for systems which implemented the SPARQL endpoint federation and hybrid query processing paradigms in Java. Only Atlas [18] implements DHT federation whereas WoDQA [2], SIHJoin [20] and LDQPS [19] implement federation over linked data (LDF). Most of the surveyed systems provides "General Public Licences" with the exception of [19] and [20] which provides "Scala" licence whereas the authors of [12] and [5] have not yet decided which licence type will hold for their tools. Five of the surveyed systems implement caching mechanisms including [32], [4], [5], [28] and [2]. Only [1] and [2] provide support for catalog/index update whereas two systems do not require this mechanism by virtue of being index/catalog-free approaches.

Table 2 summarizes the survey outcome w.r.t. different features supported by systems. Only three systems ([32], [4] and QWIDVD) claim the completeness of the results and only Avalanche [5] and DAW [28] support partial results retrieval for their implementations. Note that FedX claims result completeness when the cache that it relies on is up-to-date. Ten of the considered systems support adaptive query processing with DHT, SPLENDID, QWIDVD and DARQ being the only systems that do not support this paradigm. Only DAW [28] supports duplicate detection whereas DHT and Avalanche [5] claim to support partial duplicate detection. Granatum [12,13,17] is the only system that implements privacy and provenance. None of the considered systems implement top-k query processing or query runtime estimation.

Table 3 lists SPARQL clauses supported by the each of 14 systems. GRANATUM and QWIDVD are only two systems that support all of the query constructs used in our survey. It is important to note that most of these query constructs are based on query characteristics defined in BSBM.

4. Details of selected systems

After having given a general overview of SPARQL query federation systems, we present six SPARQL endpoints federation engines [8,25,32,1,36,21] with public implementation that were used within our experiments. We begin by presenting an overview of key concepts that underpin federated query processing and are used in the performance evaluation. We then use these key concepts to present the aforementioned six systems used in our evaluation in more detail.

4.1. Federated query processing

Given a SPARQL query $q \in Q$, where Q is a set of queries, the first step of federated SPARQL query processing is to perform *triple pattern-wise source selection* or *source selection* for short. The goal of the *triple pattern-wise source selection* is to identify the set of data sources that contain relevant results against individual triple patterns of the query [28]. We call these sources *relevant* (also called capable) for the given triple pattern. The total number of triple pattern-wise selected sources N_s is the sum of the number of sources selected for individual query triple pattern. Later, in our evaluation, we will see that N_s has a direct impact on the query execution time. The source selection information is then used to decompose q into multiple sub-queries. Each sub-query is optimized to generate an execution plan. The sub-queries are forwarded to the relevant data sources according to the optimization plan. The results of each sub-query execution are finally joined to generate the result set of q.

4.2. Overview of the selected approaches

DARQ [25] makes use of an index known as *service description* to perform *source selection*. Each service description provides a declarative description of the data available in a data source, including the corresponding SPARQL endpoint along with statistical information. The *source selection* is performed by using distinct predicates (for each data source) recorded in the index as *capabilities*. The source selection algorithm used in DARQ for a query simply matches all triple patterns against the *capabilities* of the data sources. The matching compares the predicate in a triple pattern with the predicate defined for a capability in the index. This means that DARQ is only able to answer queries with bound predicates. DARQ combines service descriptions, query rewriting mechanisms and a cost-based optimization approach to reduce the query processing time and the bandwidth usage.

SPLENDID [8] makes use of VoiD descriptions as index along with SPARQL ASK queries to perform the

Table 2

Survey outcome: System's features (**R.C.** = Results Completeness, **P.R.R.** = Partial Results Retrieval, **N.B.O.** = No Blocking Operator, **A.Q.P.** = Adaptive Query Processing, **D.D.** = Duplicate Detection, **P.B.Q.P** = Policy-based Query Planning, **Q.R.E.** = Query Runtime Estimation, **Top-K.Q.P** = Top-K query processing)

Systems	R.C.	P.R.R.	N.B.O / A.Q.P.	D. D.	P.B.Q.P	Provenance	Q.R.E	Top-K.Q.P
FedX	✓	✗	✓	✗	✗	✗	✗	✗
LHD	✗	✗	✓	✗	✗	✗	✗	✗
SPLENDID	✗	✗	✗	✗	✗	✗	✗	✗
FedSearch	✓	✗	✓	✗	✗	✗	✗	✗
GRANATUM	✗	✗	✓	✗	partial	partial	✗	✗
Avalanche	✗	✓	✓	partial	✗	✗	✗	✗
DAW	✗	✓	✓	✓	✗	✗	✗	✗
ANAPSID	✗	✗	✓	✗	✗	✗	✗	✗
ADERIS	✗	✗	✓	✗	✗	✗	✗	✗
DARQ	✗	✗	✗	✗	✗	✗	✗	✗
LDQPS	✗	✗	✓	✗	✗	✗	✗	✗
SIHJoin	✗	✗	✓	✗	✗	✗	✗	✗
WoDQA	✓	✗	✗	✗	✗	✗	✗	✗
Atlas	✗	✗	✗	partial	✗	✗	✗	✗

Table 3

Survey outcome: System's Support for SPARQL Query Constructs (QP=Query Predicates, QS=Query Subjects)

SPARQL Cluase	FedX	Atlas	LHD	SPLENDID	FedSearch	GRANATUM	Avalanche	DAW	LDQPS	SIHJoin	ANAPSID	ADERIS	QWIDVD	DARQ
SERVICE	✓	✗	✗	✗	✓	✓	✓	✓	✗	✗	✓	✗	✓	✗
FILTER	✓	✓	✓	✓	✓	✓	✗	✓	✗	✗	✓	✓	✓	✓
Unbound QP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
Unbound QS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
OPTIONAL	✓	✗	✓	✓	✓	✓	✗	✓	✗	✗	✓	✗	✓	✓
DISTINCT	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓
ORDER BY	✓	✗	✓	✓	✓	✓	✗	✓	✗	✗	✓	✗	✓	✓
UNION	✓	✓	✓	✓	✓	✓	✗	✓	✗	✗	✓	✗	✓	✓
NEGATION	✓	✗	✓	✓	✓	✓	✗	✓	✗	✗	✗	✗	✓	✓
REGEX	✓	✗	✓	✗	✓	✓	✗	✓	✗	✗	✓	✓	✓	✓
LIMIT	✓	✗	✓	✓	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓
CONSTRUCT	✓	✗	✓	✗	✓	✓	✗	✓	✗	✗	✗	✗	✓	✗
DESCRIBE	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗
ASK	✓	✗	✓	✗	✓	✓	✗	✓	✗	✗	✗	✗	✓	✗

source selection step. A SPARQL ASK query is used when any of the subject or object of the triple pattern is bound. This query is forwarded to all of the data sources and those sources which pass the SPARQL ASK test are selected. A dynamic programming strategy [34] is used to optimize the join order of SPARQL basic graph patterns.

FedX [32] is an index-free SPARQL query federation system. The source selection relies completely on SPARQL ASK queries and a cache. The cache is used to store recent SPARQL ASK operations for relevant data source selection. As shown by our evaluation, the use of this cache greatly reduces the source selection and query execution time.

The publicly available implementation of LHD [36] only makes use of the VoiD descriptions to perform source selection. The source selection algorithm is similar to DARQ. However, it also supports query triple patterns with unbound predicates. In such cases, LHD simply selects all of the available data sources as relevant. This strategy often overestimates the number of capable sources and can thus lead to high overall runtimes. LHD performs a pipeline hash join to integrate sub-queries in parallel.

ANAPSID [1] is an adaptive query engine that adapts its query execution schedulers to the data availability and runtime conditions of SPARQL endpoints. This framework provides physical SPARQL operators that detect when a source becomes blocked or data traffic is bursty. The operators produce results as quickly as data arrives from the sources. ANAPSID makes use of both a catalog and ASK queries along with heuristics defined in [22] to perform the source selection step. This heuristic-based source selection can greatly reduce the total number of triple pattern-wise selected sources.

Finally, ADERIS [21] is an index-only approach for adaptive integration of data from multiple SPARQL endpoints. The source selection algorithm is similar

to DARQ's. However, this framework also selects all of the available data sources for triple patterns with unbound predicates. ADERIS does not support several SPARQL 1.0 clauses such as UNION and OPTIONAL. For the data integration, the framework implements the pipelined index nested loop join operator.

In the next section, we describe some of the variables that may impact the performance of the federated SPARQL query engines.

5. Performance Variables

Table 4 shows some of the variables that may impact the behaviour of federated SPARQL query engines. According to [22], these variables can be grouped into two categories (i.e., independent and dependent variables) that affect the overall performance of federated query SPARQL engines. Dependent (also called observed) variables are usually the performance metrics and are normally influenced by independent variables.

Independent variables can be grouped into four dimensions: query, data, platform, and endpoint [22]. The query dimension includes:

- the type of query (star, path, hybrid [28]),
- the number of basic graph patterns,
- the instantiations (bound/unbound) of tuples (subject, predicate, object) of the query triple pattern,
- the selectivity of the joins between triple patterns,
- the query result set size, and use of different SPARQL clauses along with general predicates such as `rdf:type`, `owl:sameAs`.

The data dimension comprise of:

- the dataset size, its type of partition (horizontal, vertical, hybrid), and
- the data frequency distribution (e.g., number of subject, predicates and objects) etc.

The platform dimension consist of:

- use of cache,
- number of processor, and
- amount of RAM available.

The following parameters belong to the endpoints dimension:

- the number of endpoints used in the federation and their types (e.g., Fuseki, Sesame, Virtuoso etc., and single vs. clustered server),
- the relationship between the number of instances, graphs and endpoints of the systems used during the evaluation, and
- network latency (in case of live SPARQL endpoints) and different endpoint configuration parameters such as answer size limit, maximum resultset size etc.

In our evaluation, we measured all of the five dependent variables reported in Table 4. Most of the query (an independent variable) parameters are covered by using the complete query set of both FedX and SP^{2}Bench. However, as pointed in [22], the join selectivity cannot be fully covered due to the limitations of both FedX and SP^{2}Bench. In data parameters, the data set size cannot be fully explored in the existing SPARQL query federation benchmarks. We used horizontal partitioning and mirrored a highly distributed environment to test the selected federation systems for their parallel processing capabilities. W.r.t. platform parameters, the effect of using a cache is measured. As shown in the experiments section, the use of a cache (especially in FedX) has the potential of greatly improving the query runtime of federation systems. The amount of available RAM is more important when dealing with queries with large intermediate results, which are not given in the benchmarks at hand. The number of processors used is an important dimension to be considered in future SPARQL query federation engines. The endpoint parameters did not play a major role in our study because we used a dedicated local area network to avoid network delays. An evaluation on live SPARQL endpoints with network delay will be considered in future work. We used Virtuoso (version 20120802) SPARQL endpoints with no limit of answer size and query execution timeout.

While the dependent variables *source selection time*, *query runtime*, and *answer completeness* are already highlighted in [22], we also measured the *total number of data sources selected* and *total number of SPARQL ASK requests* used during the source selection. Section 6 shows that both of these additional variables have a significant impact on the performance of federated SPARQL query engines. For example, an overestimation of the capable sources can lead through an increase of the overall runtime due to (1) increased network traffic and (2) unnecessary intermediate results which are excluded after performing all the joins between the query triple patterns. On the other hand, the smaller the number of SPARQL ASK requests used during the source selection, the smaller the source selection time and vice versa. Further details of the depended and independent variables can be found at [22].

Table 4
Variables that impact the behavior of SPARQL federated. (#ASK = Total number of SPARQL ASK requests used during source selection, #TP= total triple pattern-wise sources selected)

Independent Variables		Dependent/Observed Variables				
		#ASK	#TP Sources	Source Selection Time	Query Runtime	Answer Completeness
Query	query plan shape	✓	✓	✓	✓	✓
	#basic triple patterns	✓	✓	✓	✓	✓
	#instantiations and their position	✓	✓	✓	✓	x
	join selectivity	x	x	x	✓	x
	#intermediate results	x	x	x	✓	x
	answer size	x	x	x	✓	x
	usage of query language expressivity	✓	✓	✓	✓	x
	#general predicates	✓	✓	✓	✓	✓
Data	dataset size	x	x	x	✓	x
	data frequency distribution	x	x	x	✓	x
	type of partitioning	✓	✓	✓	✓	✓
	data endpoint distribution	✓	✓	✓	✓	✓
Platform	cache on/off	✓	✓	✓	✓	x
	RAM available	x	x	✓	✓	x
	#processors	x	x	✓	✓	x
Endpoints	#endpoints	✓	✓	✓	✓	✓
	endpoint type	x	x	✓	✓	x
	relation graph/endpoint/instance	x	x	x	✓	✓
	network latency	x	x	✓	✓	✓
	initial delay	x	x	✓	✓	x
	message size	x	x	x	✓	x
	transfer distribution	x	x	✓	✓	✓
	answer size limit	x	x	x	✓	✓
	timeout	x	x	x	✓	✓

6. Evaluation

In this section we present the data and hardware used in our evaluation. Moreover, we explain the key metrics underlying our experiments as well as the corresponding results.

6.1. Experimental setup

We used two settings to evaluate the selected federation systems. Within the first evaluation, we used the query execution time as central evaluation parameter and made use of the FedBench [30] federated SPARQL querying benchmark. In the second evaluation, we extended both FedBench and SP^2Bench to simulate a highly federated environment. Here, we focused especially on analyzing the effect of data partitioning on the performance of federation systems. We call this extension *SlicedBench* as we created slices of each original datasets and distributed them among data sources. All of the selected performance metrics (explained in Section 6.2) remained the same for both evaluation frameworks. We used the most recent versions (at the time at which the evaluation was carried out), i.e., FedX2.0 and ANAPSID (December 2013 version). The remaining systems has no versions. All experiments were carried out on a system with a 2.60 GHz i5 processor, 4GB RAM and 500GB hard disk. For systems with Java implementation, we used Eclipse with default settings, i.e., Java Virtual Machine (JVM) initial memory allocation pool (Xms) size of 40MB and the maximum memory allocation pool (Xmx) size of 512MB. The permanents generation (MaxPermSize) which defines the memory allocated to keep compiled class files was also set to default size of 256MB. To minimise the network latency we used a dedicated local network. We conducted our experiments on local copies of Virtuoso (version 20120802) SPARQL endpoints with number of buffers 1,360,000, maximum dirty buffers 1,000,000, number of server threads 20, result set maximum rows 100,000, and maximum query execution time of 60 seconds. A separate physical virtuoso server was created

Table 5
System's specifications hosting SPARQL endpoints

EP	CPU(GHz)	RAM	Hard Disk
1	2.2, i3	4GB	300 GB
2	2.9, i7	16 GB	256 GB SSD
3	2.6, i5	4 GB	150 GB
4	2.53, i5	4 GB	300 GB
5	2.3, i5	4 GB	500 GB
6	2.53, i5	4 GB	300 GB
7	2.9, i7	8 GB	450 GB
8	2.6, i5	8 GB	400 GB
9	2.6, i5	8 GB	400 GB
10	2.9, i7	16 GB	500 GB

for each dataset. The specification of the machines hosting the virtuoso SPARQL endpoints used in both evaluations is given in Table 5. We executed each query 10 times and present the average values in the results. The source selection time (ref. Section 6.3.4) and query runtime (ref. Section 6.3.5) was calculated using the function `System.currentTimeMillis()` (for Java system implementations) and function `time()` (for Python implementations). The results of the `time()` was converted from seconds to milliseconds. The query runtime was calculated once all the results are retrieved and the time out was set to 30 minutes. Furthermore, the query runtime results were analyzed statistically using Wilcoxon signed rank test. We chose this test because it is parameter-free and does not assume a particular error distribution in the data like a t-test does. For all the significance tests, we set the p-value to 0.05.

All of the data used in both evaluations along with the portable virtuoso SPARQL endpoints can be downloaded from the project website[14].

6.1.1. First setting: FedBench

FedBench is commonly used to evaluate performance of the SPARQL query federation systems [32,8,22,28]. The benchmark is explicitly designed to represent SPARQL query federation on a real-world datasets. The datasets can be varied according to several dimensions such as size, diversity and number of interlinks. The benchmark queries resemble typical requests on these datasets and their structure ranges from simple star [28] and chain queries to complex graph patterns. The details about the FedBench datasets used in our evaluation along with some statistical information are given in Table 6.

The queries included in FedBench are divided into three categories: Cross-domain (CD), Life Sciences (LS), Linked Data (LD), and SP^{2}Bench queries. The distribution of the queries along with the result set sizes are given in Table 7. Details on the datasets and various advanced statistics are provided at the FedBench project page[15].

In this evaluation setting, we selected all queries from CD, LS, and LD, thus performing (to the best of our knowledge) the first evaluation of SPARQL query federation systems on the complete benchmark data of FedBench. It is important to note that SP^{2}Bench was designed with the main goal of evaluating query engines that access data kept in a single repository. Thus, the complete query is answered by a single data set. However, a federated query is one which collects results from multiple data sets. Due to this reason we did not include the SP^{2}Bench queries in our first evaluation. We have included all these queries into our SlicedBench because the data is distributed in 10 different data sets and each SP^{2}Bench query span over more than one data set, thus full-filling the criteria of federated query.

6.1.2. Second setting: Sliced Bench

As pointed out in [23] the data partitioning can affect the overall performance of SPARQL query federation engines. To quantify this effect, we created 10 slices of each of the 10 datasets given in Table 6 and distributed this data across 10 local virtuoso SPARQL endpoints (one slice per SPARQL endpoint). Thus, every SPARQL endpoint contained one slice from each of the 10 datasets. This creates a highly fragmented data environment where a federated query possibly had to collect data from all of the 10 SPARQL endpoints. This characteristic of the benchmark stands in contrast to FedBench where the data is not highly fragmented. Moreover, each of the SPARQL endpoint contained a comparable amount of triples (load balancing). To facilitate the distribution of the data, we used the Slice Generator tool employed in [28]. This tool allows setting a discrepancy across the slices, where the *discrepancy* is defined as the difference (in terms of number of triples) between the largest and smallest slice:

$$discrepancy = \max_{1 \leq i \leq M} |S_i| - \min_{1 \leq j \leq M} |S_j|, \quad (1)$$

where S_i stands for the ith slice. The dataset D is partitioned randomly among the slices in a way that

[14] `https://code.google.com/p/fed-eval/`

[15] `http://code.google.com/p/fbench/`

Table 6
Datasets statistics used in our benchmarks. (*only used in SlicedBench)

Collection	Dataset	version	#triples	#subjects	#predicates	#objects	#types	#links
Cross Domain	DBpedia subset	3.5.1	43.6M	9.50M	1.06k	13.6M	248	61.5k
	GeoNames	2010-10-06	108M	7.48M	26	35.8M	1	118k
	LinkedMDB	2010-01-19	6.15M	694k	222	2.05M	53	63.1k
	Jamendo	2010-11-25	1.05M	336k	26	441k	11	1.7k
	New York Times	2010-01-13	335k	21.7k	36	192k	2	31.7k
	SW Dog Food	2010-11-25	104k	12.0k	118	37.5k	103	1.6k
Life Sciences	KEGG	2010-11-25	1.09M	34.3k	21	939k	4	30k
	ChEBI	2010-11-25	7.33M	50.5k	28	772k	1	-
	Drugbank	2010-11-25	767k	19.7k	119	276k	8	9.5k
	DBpedia subset 3.5.1	43.6M	9.50M	1.06k	13.6M	248	61.5k	61.5k
SP^{2}Bench*	SP^{2}Bench 10M	v1.01	10M	1.7M	77	5.4M	12	-

Table 7
Query characteristics, (#T = Total number of Triple patterns, #Res = Total number of query results, *only used in SlicedBench, **OP**erators: **A**nd (“.”), **U**nion, **F**ilter, **O**ptional; **Struc**ture: **S**tar, **C**hain, **H**ybrid).

Linked Data (LD)					Cross Domain (CD)					Life Science (LS)					SP^{2}Bench*				
Q	#T	#Res	Op	Struct		#T	#Res	Op	Struct	Q	#T	#Res	Op	Struct	Q	#T	#Res	Op	Struc
1	3	309	A	C	1	3	90	AU	S	1	2	1159	U	-	1	3	1	A	S
2	3	185	A	C	2	3	1	A	S	2	3	333	AU	-	2	10	>500k	AO	S
3	4	162	A	C	3	5	2	A	H	3	5	9054	A	H	3a	3	27789	AF	S
4	5	50	A	C	4	5	1	A	C	4	7	3	A	H	4	8	>40M	AF	C
5	3	10	A	S	5	4	2	A	C	5	6	393	A	H	5b	5	>300k	AF	C
6	5	11	A	H	6	4	11	A	C	6	5	28	A	H	6	9	>700k	AFO	H
7	2	1024	A	S	7	4	1	A	C	7	5	144	AFO	H	7	12	>2k	AFO	H
8	5	22	A	H											8	10	493	AFU	H
9	3	1	A	C											9	4	4	AU	-
10	3	3	A	C											10	1	656		-
11	5	239	A	S											11	1	10	-	-

$\sum_i |S_i| = |D|$ and $\forall i \forall j \; i \neq j \rightarrow ||S_i| - |S_j|| \leq discrepancy$.

This tool generate slices based on horizental partioning of the data. Table 8 shows the discrepancy values used for slice generation for each of the 10 datasets. Our discrepancy value varies with the size of the dataset. For the query runtime evaluation, we selected all of the queries both from FedBench and SP^{2}Bench given in Table 7: the reason for this selection was to cover majority of the SPARQL query clauses and types along with variable results size (from 1 to 40 million). For each of the CD, LS, and LD queries used in SlicedBench, the number of results remained the same as given in Table 7. Analogously to FedBench, each of the SlicedBench data source is a virtuoso SPARQL endpoint.

Table 8
Dataset slices used in SlicedBench

Collection	#Slices	Discrepancy
DBpedia subset 3.5.1	10	280,000
GeoNames	10	600,000
LinkedMDB	10	100,000
Jamendo	10	30,000
New York Times	10	700
SW Dog Food	10	200
KEGG	10	35,000
ChEBI	10	50,000
Drugbank	10	25,000
SP^{2}Bench	10	150,000

6.2. Evaluation criteria

We selected four metrics for our evaluation: (1) *total triple pattern-wise sources selected*, (2) total number of SPARQL ASK requests used during source selection,

SELECT o1, o2
WHERE
{
?s p1 ?o1. //tp_1
?s p2 ?o2. //tp_2
}
	D_1 = {s_1, s_2, s_4 }
	D_2 = {s_1, s_3, s_5, s_9 }
	$\|D_1\|$ = 3
	$\|D_2\|$ = 4
	Total TP. Sources = $\|D_1\|$ + $\|D_2\|$
	= 7

Fig. 1. Total triple pattern-wise selected sources example. (TP. = triple pattern-wise selected)

(3) *answer completeness*(4) *source selection time* (i.e. the time taken by the process in the first metric), and (5) *query execution time*.

The total number of triple pattern-wise selected sources for a query is calculated as follows: Let $D_i = \{s_1, s_2 \dots s_m\}$ be the set of sources capable of answering a triple pattern tp_i and M is the total number of available (physical) sources. Then, for a query q with n triple patterns, $\{tp_1, tp_2, \dots tp_n\}$, the total number of triple pattern-wise sources is the sum of the magnitude ($|D_i|$) of capable sources set for individual triple patterns. An example of the triple pattern-wise source selection is given in Figure 1 considering there are three sources capable of answering the first triple pattern tp_1 and four sources capable of answering tp_2 summing up to a total triple pattern-wise selected sources equal to seven.

An overestimation of triple pattern-wise selected sources increases the source selection time and thus the the query execution time. Furthermore, such an overestimation increases the number of irrelevant results which are excluded after joining the results of the different sources, therewith increasing both the network traffic and query execution time. In the next section we explain how such overestimations occur in the selected approaches.

6.3. Experimental results

6.3.1. Triple pattern-wise selected sources

Table 9 shows the total number of triple pattern-wise sources (TP sources for short) selected by each approach both for the FedBench and SlicedBench queries. ANAPSID is the most accurate system in terms of TP sources followed by both FedX and SPLENDID whereas similar results are achieved by the other three systems, i.e., LHD, DARQ, and ADERIS. Both FedX and SPLENDID select the optimal number of TP sources for individual query triple patterns. This is because both make use of ASK queries when any of the subject or object is bound in a triple pattern. However, they do not consider whether a source can actually contribute results after performing a join between results with other query triple patterns. Therefore, both can overestimate the set of capable sources that can actually contribute results. ANAPSID uses a catalog and ASK queries along with heuristics [22] about triple pattern joins to reduce the overestimation of sources. LHD (the publicly available version), DARQ, and ADERIS are index-only approaches and do not use SPARQL ASK queries when any of the subject or object is bound. Consequently, these three approaches tend to overestimate the TP sources per individual triple pattern. It is important to note that DARQ does not support queries where any of the predicates in a triple pattern is unbound (e.g., CD1, LS2) and ADERIS does not support queries which feature FILTER or UNION clauses (e.g., CD1, LS1, LS2, LS7). In case of triple patterns with unbound predicates (such as CD1, LS2) both LHD and ADERIS simply select all of the available sources as relevant. This overestimation can significantly increase the overall query execution time.

The effect overestimation can be clearly seen by taking a fine-granular look at how the different systems process FedBench query CD3 given in Listing 1. The optimal number of TP sources for this query is 5. This query has a total of five triple patterns. To process this query, FedX sends a SPARQL ASK query to all of the 10 benchmark SPARQL endpoints for each of the triple pattern summing up to a total of 50 (5*10) SPARQL ASK operations. As a result of these operations, only one source is selected for each of the first four triple pattern while eight sources are selected for last one, summing up to a total of 12 TP sources. SPLENDID utilizes its index and ASK queries for the first three and index-only for last two triple pattern to select exactly the same number of sources selected by FedX. LHD, ADERIS, and DARQ only makes use of predicate lookups in their catalogs to select nine sources for the first, one source each for the second, third, fourth, and eighth for the last triple pattern summing up to a total of 20 TP sources. The later three approaches overestimate the number of sources for first triple pattern by 8 sources. This is due to the predicate `rdf:type` being likely to be used in all of RDF datasets. However, triples with `rdf:type` as predicate and the bound object `dbp:President` are only contained in the DBpedia subset of FedBench. Thus, the only relevant data source for the first triple pattern is DBpedia subset. Interestingly, even FedX and SPLENDID overestimate the number of data sources that can contribute for the last triple pattern. There are

Table 9
Comparison of triple pattern-wise total number of sources selected for FedBench and SlicedBench. NS stands for "not supported", RE for "runtime error", SPL for SPLENDID, ANA for ANAPSID and ADE for ADERIS. Key results are in **bold**.

FedBench							SlicedBench						
Query	FedX	SPL	LHD	DARQ	ANA	ADE	Query	FedX	SPL	LHD	DARQ	ANA	ADE
CD1	11	11	**28**	NS	3	NS	CD1	17	17	**30**	NS	8	NS
CD2	3	3	10	10	3	10	CD2	12	12	24	24	12	24
CD3	**12**	**12**	**20**	**20**	**5**	**20**	CD3	31	31	38	38	31	38
CD4	19	19	20	20	5	20	CD4	32	32	34	34	32	34
CD5	11	11	11	11	4	11	CD5	19	19	19	19	9	19
CD6	**9**	**9**	**10**	**10**	**10**	**10**	CD6	31	31	40	40	31	40
CD7	13	13	13	13	6	13	CD7	40	40	40	40	40	40
Total	**78**	**78**	**112**	**84**	**36**	**84**	**Total**	**182**	**182**	**225**	**195**	**163**	**195**
LS1	**1**	**1**	**1**	**1**	**1**	NS	LS1	3	3	3	3	3	NS
LS2	**11**	**11**	**28**	NS	**12**	NS	LS2	16	16	**30**	NS	16	NS
LS3	12	12	20	20	5	20	LS3	19	19	26	26	19	26
LS4	7	7	15	15	7	15	LS4	25	25	27	27	14	27
LS5	10	10	18	18	7	18	LS5	30	30	37	37	20	37
LS6	9	9	17	17	5	17	LS6	19	19	27	27	17	27
LS7	6	6	6	6	7	NS	LS7	13	13	13	13	13	NS
Total	**56**	**56**	**105**	**77**	**44**	**70**	**Total**	**125**	**125**	**163**	**133**	**102**	**117**
LD1	8	8	11	11	3	11	LD1	10	10	29	29	3	29
LD2	3	3	3	3	3	3	LD2	20	20	28	28	20	28
LD3	16	16	16	16	4	16	LD3	30	30	39	39	13	39
LD4	5	5	5	5	5	5	LD4	30	30	47	47	5	47
LD5	5	5	13	13	3	13	LD5	15	15	24	24	15	24
LD6	14	14	14	14	14	14	LD6	38	38	38	38	38	38
LD7	3	3	4	4	2	4	LD7	12	12	20	20	12	20
LD8	15	15	15	15	9	15	LD8	27	27	27	27	16	27
LD9	3	3	6	6	3	6	LD9	7	7	17	17	7	17
LD10	10	10	11	11	3	11	LD10	23	23	23	23	23	23
LD11	15	15	15	15	5	15	LD11	31	31	32	32	31	32
Total	**108**	**108**	**119**	**122**	**54**	**119**	**Total**	**243**	**243**	**324**	**324**	**183**	**324**
							SP2B-1	10	10	28	28	NS	28
							SP2B-2	90	90	92	92	RE	NS
							SP2B-3a	13	13	19	NS	13	19
							SP2B-4	52	52	66	66	52	66
							SP2B-5b	40	40	50	50	40	50
							SP2B-6	68	68	72	72	18	NS
							SP2B-7	100	100	104	NS	64	NS
							SP2B-8	91	91	102	102	NS	NS
							SP2B-9	40	40	40	NS	40	NS
							SP2B-10	7	7	10	NS	7	10
							SP2B-11	10	10	10	10	10	NS
							Total	**521**	**521**	**593**	**420**	**244**	**173**
Net Total	**242**	**242**	**336**	**283**	**134**	**273**	**Net Total**	**1071**	**1071**	**1305**	**1072**	**692**	**809**

```
SELECT ?president ?party ?page
WHERE {
?president rdf:type dbp:President .
?president dbp:nationality dbp:US .
?president dbp:party ?party .
?x nyt:topicPage ?page .
?x owl#sameAs ?president .
}
```

Listing 1: FedBench CD3. Prefixes are ignored for simplicity

eight FedBench datasets which contain `owl:sameAs` predicate. However, only one (i.e., New York Times) can actually contribute results after a join of the last two triple patterns is carried out. ANAPSID makes use of a catalog and SPARQL-ASK-assisted Star Shaped Group Multiple (SSGM) endpoint selection heuristic [22] to select the optimal (i.e., five) TP sources for this query. However, ANAPSID also overestimates the TP sources in some cases. For query CD6 of FedBench, ANAPSID selected a total of 10 TP sources while only 4 is the optimal sources that actually contributes to the final result set. This behavior leads us to our first insight: Optimal TP source selection is not sufficient to detect the optimal set of sources that should be queried.

In the SlicedBench results, we can clearly see the TP values are increased for each of the FedBench queries which mean a query spans more data sources, thus simulating a highly fragmented environment suitable to test the federation system for effective parallel query processing. The highest number of TP sources are reported for the second SP^2Bench query where up to a total of 92 TP sources are selected. This query contains 10 triple patterns and index-free approaches (e.g., FedX) need 100 (10*10) SPARQL ASK queries to perform the source selection operation. Using SPARQL ASK queries with no caching for such a highly federated environment can be very expensive. From the results shown in Table 9, it is noticeable that hybrid (catalog + SPARQL ASK) source selection approaches (ANAPSID, SPLENDID) perform an more accurate source selection than index/catalog-only approaches (i.e., DARQ, LHD, and ADERIS).

6.3.2. Number of SPARQL ASK requests

Table 10 shows the total number of SPARQL ASK requests used to perform source selection for each of the queries of FedBench and SlicedBench. Index-only approaches (DARQ, ADERIS, LHD) only make use of their index to perform source selection. Therefore, they do not necessitate any ASK requests to process queries. As mention before, FedX only makes use of ASK requests (along with a cache) to perform source selection. The results presented in Table 10 are for FedX(cold or first run), where the FedX cache is empty. This is basically the lower bound of the performance of FedX. For FedX(100% cached), the complete source selection is performed by using cache entries only. Hence, in that case, the number of SPARQL ASK requests is zero for each query. This is the upper bound of the performance of FedX on the data at hand. The results clearly shows that index-free (e.g., FedX) approaches can be very expensive in terms of SPARQL ASK requests used. This can greatly affect the source selection time and overall query execution time if no cache is used. For FedBench, ANAPSID is the most efficient hybrid approach in terms of SPARQL ASK requests consumed during source selection. On the other hand, SPLENDID is the most efficient hybrid approach for SlicedBench. The reason for higher number of ASK requests is due to the logic behind SSGM heuristics [22] used in ANAPSID's source selection algorithm. This heuristic computes the set of namespaces and predicates used in data sources (SPARQL endpoints in our case). For SlicedBench, all data sources are likely contains the same set of distinct predicates and namespaces (because each data source contains at least one slice from each data dump). SSGM heuristics are bound to perform poorly when faced with such a data distribution.

It is important to note that ANAPSID combines more than one triple pattern into a single SPARQL ASK query. The time required to execute these more complex SPARQL ASK operations are generally higher than SPARQL ASK queries having a single triple pattern as used in FedX and SPLENDID. Consequently, even though ANAPSID require less SPARQL ASK requests, its source selection time is greater than all other selected approaches. This behavior will be further elaborated upon in the subsequent section. Tables 9 and 10 clearly show that using SPARQL ASK queries for source selection leads to an efficient source selection in terms of TP sources selected. However, in the next section we will see that they increase both source selection and overall query runtime. A smart source selection approach should select fewer number of TP sources while using minimal number of SPARQL ASK requests.

6.3.3. Answer compeleteness

As pointed in [22], an important criterion in performance evaluation of the federated SPARQL query en-

Table 10

Comparison of number of SPARQL ASK requests used for source selection both in FedBench and SlicedBench. NS stands for "not supported", RE for "runtime error", SPL for SPLENDID, ANA for ANAPSID and ADE for ADERIS. Key results are in **bold**.

FedBench							SlicedBench						
Query	FedX	SPL	LHD	DARQ	ANA	ADE	Query	FedX	SPL	LHD	DARQ	ANA	ADE
CD1	27	27	0	NS	20	NS	CD1	30	30	0	NS	25	NS
CD2	27	18	0	0	1	0	CD2	30	20	0	0	29	0
CD3	45	18	0	0	2	0	CD3	50	20	0	0	46	0
CD4	45	9	0	0	3	0	CD4	50	10	0	0	34	0
CD5	36	9	0	0	1	0	CD5	40	10	0	0	14	0
CD6	36	9	0	0	11	0	CD6	40	10	0	0	40	0
CD7	36	9	0	0	5	0	CD7	40	10	0	0	40	0
Total	**252**	**99**	**0**	**0**	**43**	**0**	**Total**	**280**	**110**	**0**	**0**	**228**	**0**
LS1	18	0	0	0	0	NS	LS1	20	0	0	0	3	NS
LS2	27	27	0	NS	30	NS	LS2	30	30	0	NS	30	NS
LS3	45	9	0	0	13	0	LS3	50	10	0	0	30	0
LS4	63	18	0	0	1	0	LS4	70	20	0	0	15	0
LS5	54	9	0	0	4	0	LS5	60	10	0	0	27	0
LS6	45	18	0	0	13	0	LS6	50	20	0	0	26	0
LS7	45	9	0	0	2	NS	LS7	50	10	0	0	12	NS
Total	**297**	**90**	**0**	**0**	**63**	**0**	**Total**	**330**	**100**	**0**	**0**	**143**	**0**
LD1	27	9	0	0	1	0	LD1	30	10	0	0	12	0
LD2	27	9	0	0	0	0	LD2	30	10	0	0	29	0
LD3	36	9	0	0	2	0	LD3	40	10	0	0	23	0
LD4	45	18	0	0	0	0	LD4	50	20	0	0	25	0
LD5	27	18	0	0	2	0	LD5	30	20	0	0	32	0
LD6	45	9	0	0	12	0	LD6	50	10	0	0	38	0
LD7	18	9	0	0	4	0	LD7	20	10	0	0	20	0
LD8	45	9	0	0	7	0	LD8	50	10	0	0	19	0
LD9	27	18	0	0	3	0	LD9	30	20	0	0	17	0
LD10	27	9	0	0	4	0	LD10	30	10	0	0	23	0
LD11	45	9	0	0	2	0	LD11	50	10	0	0	32	0
Total	**369**	**126**	**0**	**0**	**37**	**0**	**Total**	**410**	**140**	**0**	**0**	**270**	**0**
							SP2B-1	30	20	0	0	NS	0
							SP2B-2	100	10	0	0	RE	NS
							SP2B-3a	20	10	0	NS	10	0
							SP2B-4	80	20	0	0	66	0
							SP2B-5b	50	20	0	0	50	0
							SP2B-6	90	20	0	0	37	NS
							SP2B-7	130	30	0	NS	62	NS
							SP2B-8	100	20	0	0	NS	NS
							SP2B-9	40	20	0	NS	20	NS
							SP2B-10	10	10	0	NS	10	0
							SP2B-11	10	0	0	0	10	NS
							Total	**660**	**180**	**0**	**0**	**265**	**0**
Net Total	**918**	**315**	**0**	**0**	**143**	**0**	**Net Total**	**1680**	**530**	**0**	**0**	**906**	**0**

Table 11
The queries for which system's did not retrieve complete results. The values inside bracket shows the actual results size

	CD1(90)	CD7(1)	LS1(1159)	LS2(333)	LS3(9054)	LS5(393)	LD1(309)	LD3(162)	LD9(1)	SP2B-3a(27789)	SP2B-6(>700k)
SPLENDID	-	-	-	-	-	-	308	159	-	-	-
LHD	77	-	0	322	0	0	-	-	-	-	-
ANAPSID	-	0	-	-	-	-	-	-	-	0	0
ADERIS	77	-	-	-	-	-	-	-	-	-	-
DARQ	-	-	-	-	-	-	-	-	0	-	-

gines is the result set completeness. Two or more engines are only comparable to each other if they provide the same result set for a given query. A federated engine may miss results due to various reasons including the type of source selection used, the use of an outdated cache or index, the type of network, the endpoint result size limit or even the join implementation. In our case, the sole possible reason for missing results across all six engines is the join implementation as all of the selected engines overestimate the set of capable sources (i.e., they never generate false negatives w.r.t. the capable sources), the cache, index are always up-to-date, the endpoint result size limit is greater than the query results and we used a local dedicated network with negligible network delay. Table 11 shows the queries and federated engines for which we did not receive the complete results. As an overall answer completeness evaluation, only FedX is always able to retrieve complete results. It is important to note that these results are directly connected to the answer completeness results presented in survey Table 2; which shows only FedX is able to provide complete results among the selected systems.

6.3.4. Source selection time

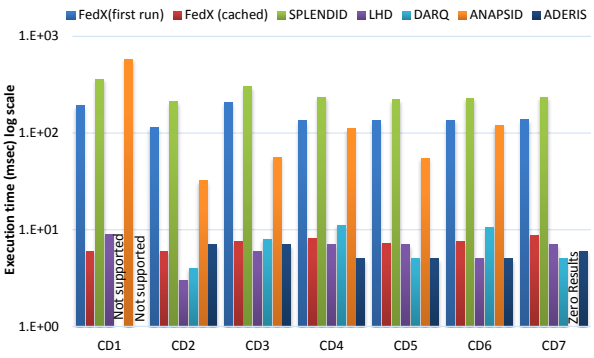
Figure 2 and Figure 3 shows the source selection time for each of the selected approach and for both FedBench and SlicedBench. Compared to the TP results, the index-only approaches require less time than the hybrid approaches even though they overestimated the TP sources in comparison with the hybrid approaches. This is due to index-only approaches not having to send any SPARQL ASK queries during the source selection process. The index being usually pre-loaded into the memory before the query execution means that the runtime the predicate look-up in index-only approaches is minimal. Consequently, we observe a trade-off between the intelligent source selection and the time required to perform this process. To reduce the costs associated with ASK operations, FedX implements a cache to store the results of the recent SPARQL ASK operations. Figure 2 shows that source selection time of FedX with cached entries is significantly smaller than FedX's first run with no cached entries.

As expected the source selection time for FedBench queries is smaller than that for SlicedBench, particularly in hybrid approaches. This is because the number of TP sources for SlicedBench queries are increased due to data partitioning. Consequently, the number of SPARQL ASK requests grows and increases the overall source selection time. As mentioned before, an overestimation of TP sources in highly federated environments can greatly increase the source selection time. For example, consider query LD4. SPLENDID selects the optimal (i.e., five) number of sources for FedBench and the source selection time is 218 ms. However, it overestimates the number of TP sources for SlicedBench by selecting 30 instead of 5 sources. As a result, the source selection time is significantly increased to 1035 ms which directly affects the overall query runtime. The effect of such overestimation is even worse in SP2B-2 and SP2B-4 queries for the SlicedBench.

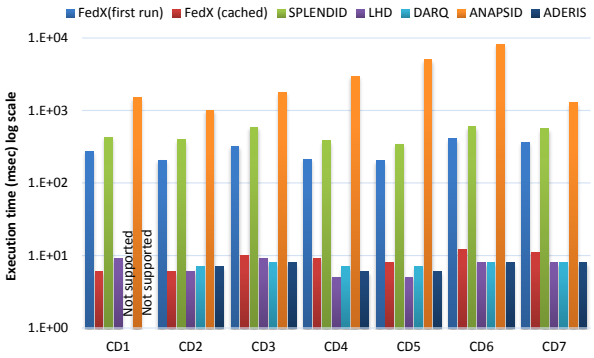
Lessons learned from the evaluation of the first three metrics is that using ASK queries for source selection leads to smart source selection in term of total TP sources selected. On the other hand, they significantly increase the overall query runtime where no caching is used. FedX makes use of an intelligent combination of parallel ASK query processing and caching to perform the source selection process. This parallel execution of SPARQL ASK queries is more time-efficient than the ASK query processing approaches implemented in both ANAPSID and SPLENDID. Nevertheless, the source selection of FedX could be improved further by using heuristics such as ANAPSID's to reduce the overestimation of TP sources.

6.3.5. Query execution time

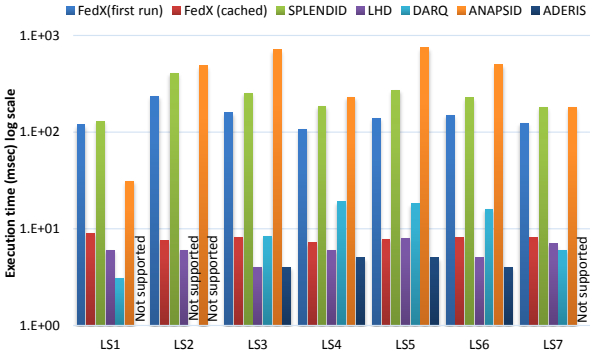
Figure 4 and Figure 5 show the query execution time for both experimental setups. The negligibly small standard deviation error bars (shown on top of each bar) indicate that the data points tend to be very close to the mean, thus suggest a high consistency of the query runtimes in most frameworks. As an overall query execution time evaluation, FedX(cached) outperforms all



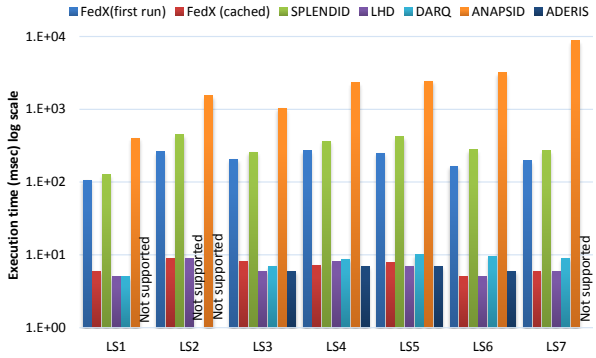
(a) FedBench: CD queries



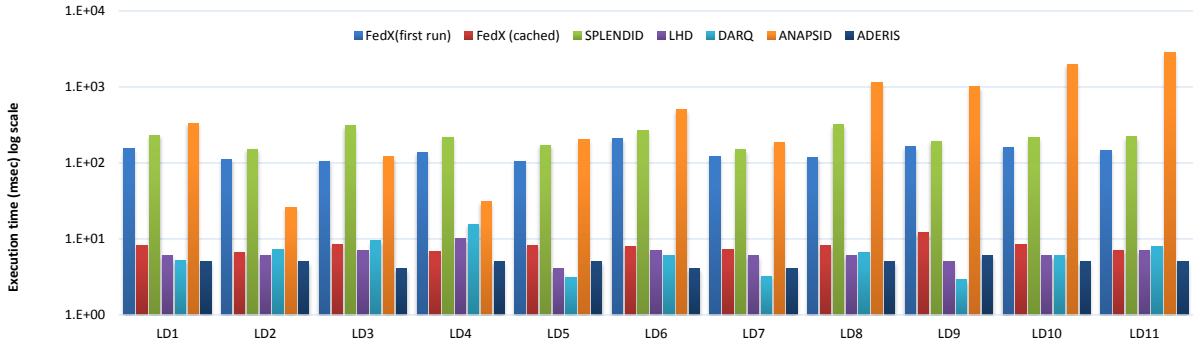
(b) SlicedBench: CD queries



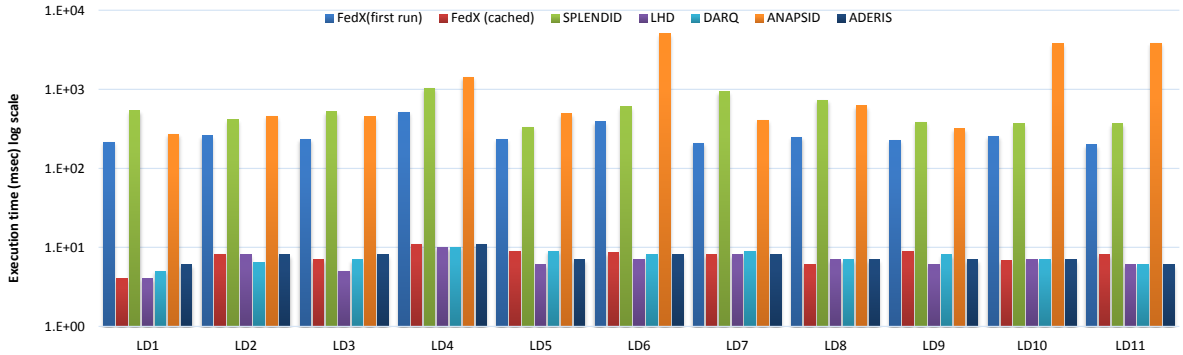
(c) FedBench: LS queries



(d) SlicedBench: LS queries



(e) FedBench: LD queries



(f) SlicedBench: LD queries

Fig. 2. Comparison of source selection time for FedBench and SlicedBench.

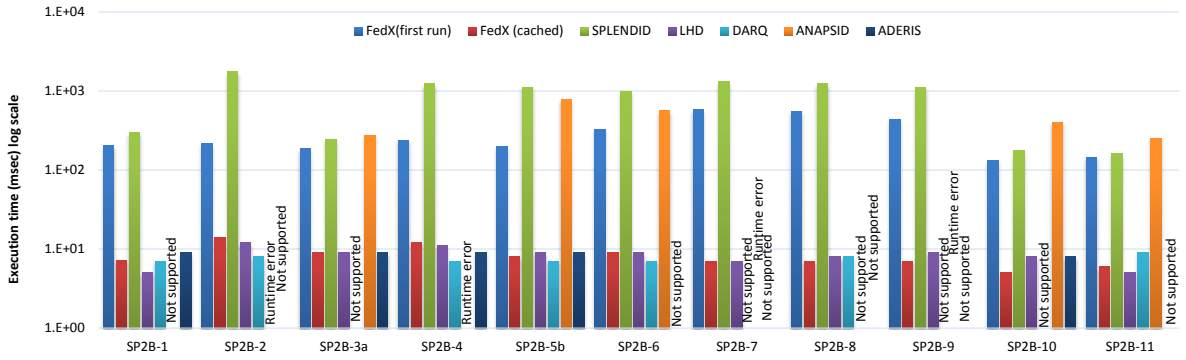


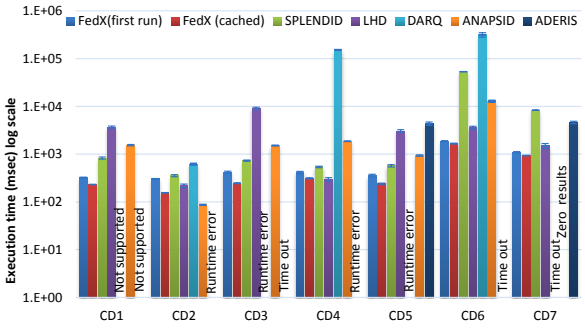
Fig. 3. Comparison of source selection time for SP^{2}Bench queries in SlicedBench.

of the remaining approaches in majority of the queries. FedX(cached) is followed by FedX(first run) which is further followed by LHD, SPLENDID, ANAPSID, ADERIS, and DARQ. Deciding between DARQ and ADERIS is not trivial because the latter does not produce results for most of the queries. The exact number of queries by which one system is better than other is given in the next section (ref. Section 7.1). Furthermore, the number of queries by which one system significantly outperform other (using wilcoxon signed rank test) is also given in the next section.

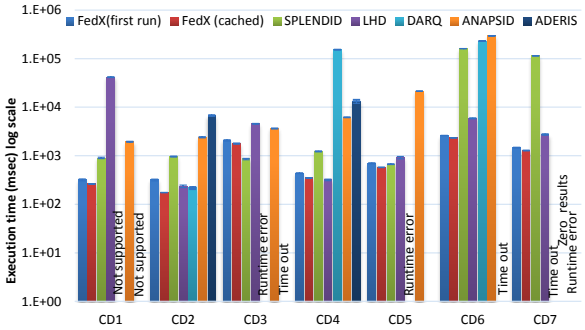
Interestingly, while ANAPSID ranks first (among the selected systems) in terms of triple pattern-wise sources selected results, it ranks fourth in terms of query execution performance. There are a couple of reason for this: (1) ANAPSID does not make use of cache. As a result, it spends more time (8ms for FedX and 1265 ms for ANAPSID on average over both setups) performing source selection, which worsens its query execution time and (2) Bushy tress (used in ANAPSID) only perform better than left deep trees (used in FedX) when the queries are more complex and triple patterns joins are more selective [15,3]. However, the FedBench queries (excluding SP2Bench) are not very selective and are rather simple, e.g., triple patterns in a query ranges from 2 to 7. In addition, the query result set sizes are small (10 queries whose resultset size smaller than 16) and the average query execution is small (about 3 seconds on average for FedX over both setups). The SP2Bench queries are more complex and the resultset sizes are large. However, the selected systems were not able to execute majority of the SP2Bench queries. It would be interesting to compare these systems on more complex and Big Data benchmark. The use of a cache improves FedX's performance by 10.5% in the average query execution for FedBech and 4.14% in SlicedBench.

The effect of the overestimation of the TP sources on query execution can be observed on the majority of the queries for different systems. For instance, for FedBench's LD4 query SPLENDID selects the optimal number of TP sources (i.e., five) and the query execution time is 318 ms of which 218 ms are used for selecting sources. For SlicedBench, SPLENDID overestimates the TP sources by 25 (i.e., selects 30 instead of 5 sources), resulting in a query execution of 10693 ms, of which 1035 ms are spent in the source selection process. Consequently, the pure query execution time of this query is only 100 ms for FedBench (318-218) and 9659 ms (10693-1035) for SlicedBench. This means that an overestimation of TP sources does not only increase the source selection time but also produces results which are excluded after performing join operation between query triple patterns. These retrieval of irrelevant results increases the network traffic and thwarts the query execution plan. For example, both FedX and SPLENDID considered 285,412 irrelevant triples due to the overestimation of 8 TP sources only for $owl:sameAs$ predicate in CD3 of FedBench. Another example of TP source overestimation can seen in CD1, LS2. LHD's overestimation of TP sources on SlicedBench (e.g., 22 for CD1, 14 for LS2) leads to its query execution time jumping from 3,670.9 ms to 41,586.3 ms for CD1 and 427 ms to 34418.3 ms for LS2.

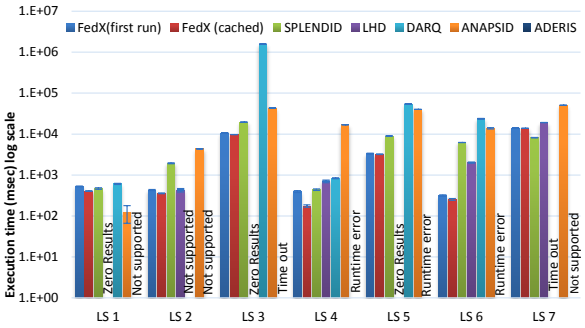
In queries such as CD4, CD6, LS3, LD11 and SP2B-11 we observe that the query execution time for DARQ is more than 2 minutes. In some cases, it even reaches the 30 minute timeout used in our experiments. The reason for this behaviour is that the simple nested loop join it implements overfloods SPARQL endpoints by submitting too many endpoint requests. FedX overcomes this problem by using a block nested loop join where the number of endpoints requests are dependent upon the block size. Furthermore, we can see that many systems



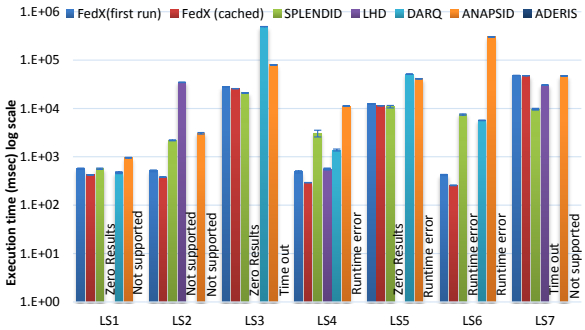
(a) FedBench: CD queries



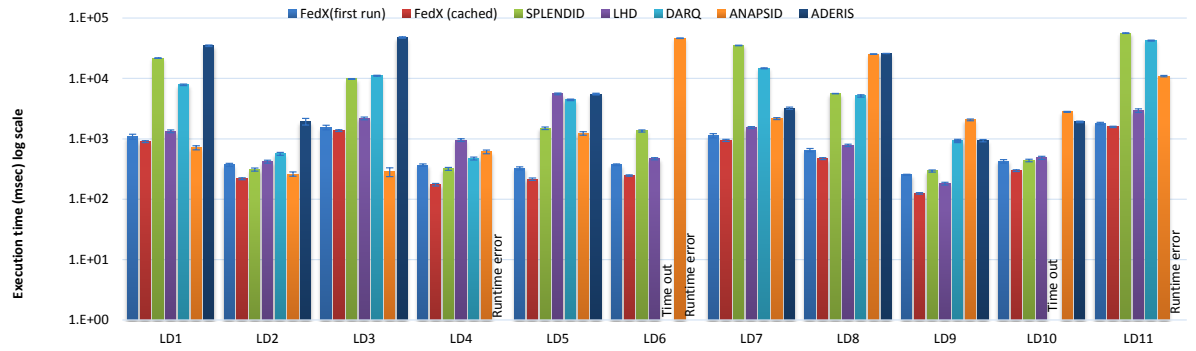
(b) SlicedBench: CD queries



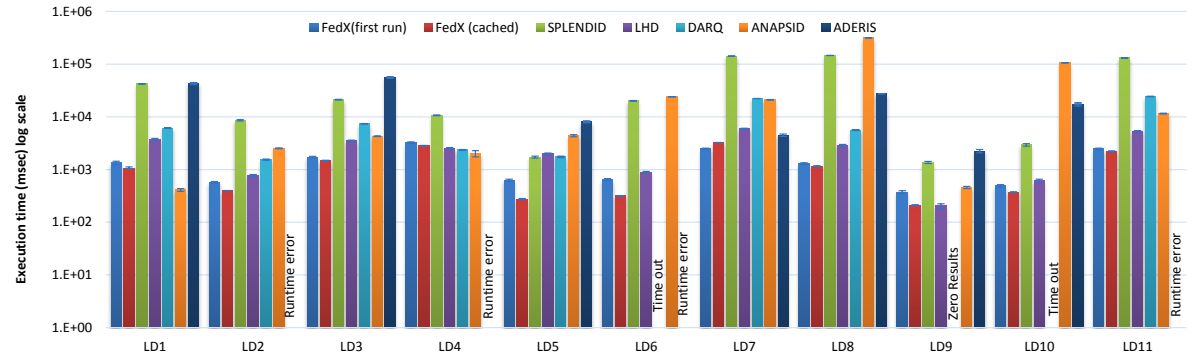
(c) FedBench: LS queries



(d) SlicedBench: LS queries



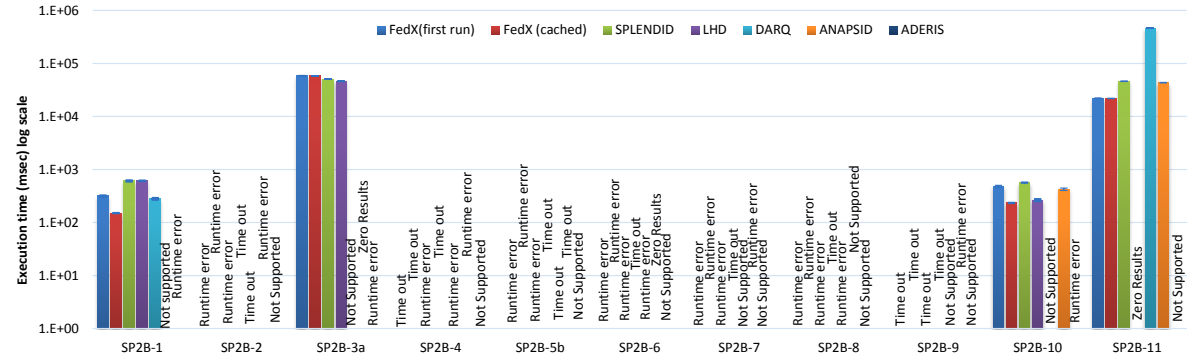
(e) FedBench: LD queries



(f) SlicedBench: LD queries

Fig. 4. Comparison of query execution time for FedBench and SlicedBench

Fig. 5. Comparison of query execution time forSlicedBench: SP^{2}Bench queries



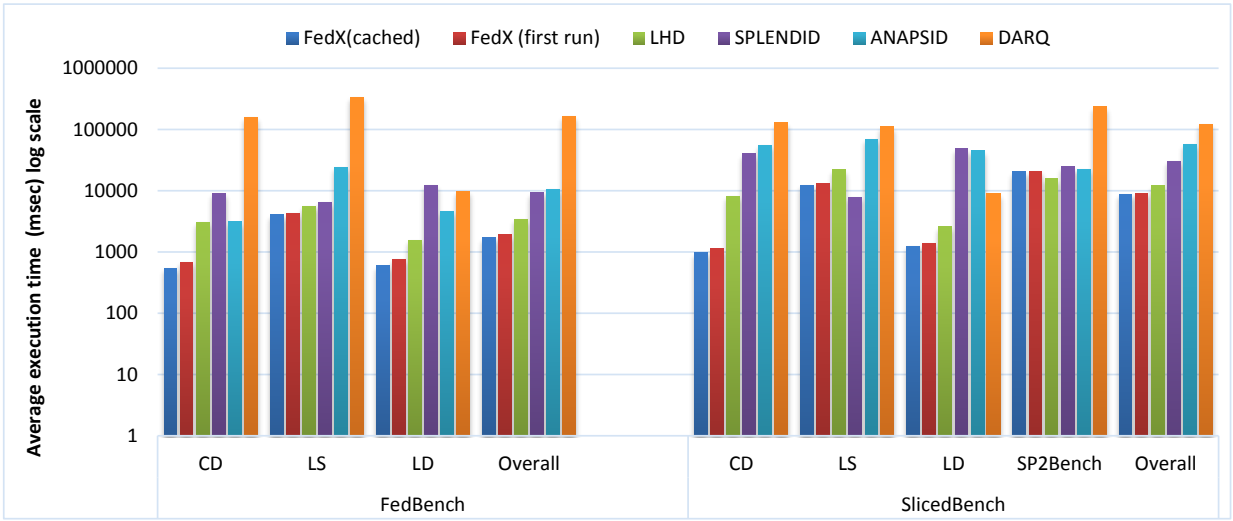


Fig. 6. Overall performance evaluation (ms)

do not produce results for SP^{2}Bench queries. A possible reason for this is the fact that SP^{2}Bench queries contain up to 10 triple patterns with different SPARQL clauses such as DISTINCT, ORDER BY, and complex FILTERS.

6.3.6. Overall performance evaluation

The comparison of the overall performance of each approach is summarised in Figure 6, where we show the average query execution time for the queries in CD, LS, LD, and SP^{2}Bench sub-groups. As an overall performance evaluation based on FedBench, FedX(cached) outperformed FedX(first run) on all of the 25 queries. FedX(first run) in turn outperformed LHD on 17 out of 22 commonly supported queries (LHD retrieve zero results for three queries). LHD is better than SPLENDID in 13 out of 22 comparable queries. SPLENDID outperformed ANAPSID in 15 out of 24 queries while ANAPSID outperforms DARQ in 16 out of 22 commonly supported queries. For SlicedBench, FedX(cached) outperformed FedX(first run) in 29 out of 36 comparable queries. In turn FedX(first run) outperformed LHD in 17 out of 24 queries. LHD is better than SPLENDID in 17 out of 24 comparable queries. SPLENDID outperformed ANAPSID in 17 out of 26 which in turn outperformed DARQ in 12 out of 20 commonly supported queries. No results were retrieved for majority of the queries in case of ADERIS, hence not included to this section. All of the above improvements are significant based on wilcoxon signed ranked test with significance level set to 0.05.

7. Discussion

The subsequent discussion of our findings can be divided into two main categories.

7.1. Effect of the source selection time

To the best of our knowledge, the effect of the source selection runtime has not been considered in SPARQL query federation system evaluations [8,1,32,26,22] so far. However, after analysing all of the results presented above, we noticed that this metric greatly affects the overall query execution time. To show this effect, we compared the pure query execution time (excluding source selection time). To calculate the pure query execution time, we simply subtracted the source selection time from the overall query execution and plot the execution time in Figure 7.

We can see that the overall query execution time (including source selection given in Figure 4) of SPLENDID is better than FedX(cached) in only one out of the 25 FedBench queries. However, Figure 7 suggests that SPLENDID is better in 8 out of the 25 queries in terms of the pure query execution time. This means that SPLENDID is slower than FedX (cached) in 33% of the queries only due to the source selection process. Furthermore, our results also suggest that the use of SPARQL ASK queries for source selection is expensive without caching. On average, SPLENDID's source selection time is 235 ms for FedBench and 591 ms in case of SlicedBench. On the other hand, FedX (cached)'s source selection time is 8ms for both FedBench and SlicedBench. ANAPSID average source selection time for FedBench is 507 ms and 2014 ms for SlicedBench which is one of the reason of ANAPSID poor performance as compare to FedX (cached).

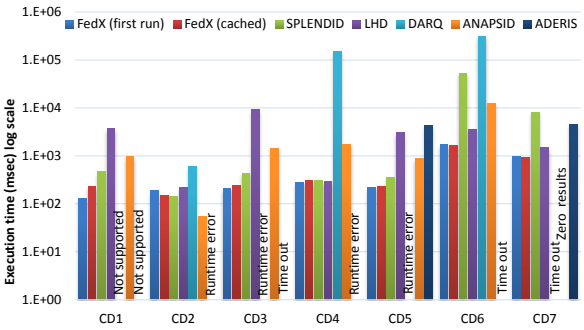
7.2. Effect of the data partitioning

In our SlicedBench experiments, we extended FedBench to test the federation systems behaviour in highly federated data environment. This extension can also be utilized to test the capability of parallel execution of queries in SPARQL endpoint federation system. To show the effect of data partioning, we calculated the average for the query execution time of LD, CD, and LS for both the benchmarks and compared the effect on each of the selected approach. The performance of FedX(cached) and DARQ is improved with partitioning while the performance of FedX(first run), SPLENDID, ANAPSID, and LHD is reduced. As an overall evaluation result, FedX(first run)'s performance is reduced by 214%, FedX(cached)'s is reduced 199%, SPLENDID's is reduced by 227%, LHD's is reduced by 293%, ANAPSID's is reduced by 382%, and interestingly DARQ's is improved by 36%. This results suggest that FedX is the best system in terms of parallel execution of queires, followed by SPLENDID, LHD, and ANAPSID. The performance improvement for DARQ occurs due to the fact that the overflooding of endpoints with too many nested loop requests to a particular endpoint is now reduced. This reduction is due to the different distribution of the relevant results among many SPARQL endpoints. One of the reasons for the performance reduction in LHD is its significant overestimation of TP sources in SlicedBench. The reduction of both SPLENDID's and ANAPSID's performance is due to an increase in ASK operations in SlicedBench and due to the increase in triple pattern-wise selected sources which greatly affects the overall performance of the systems when no cache used.

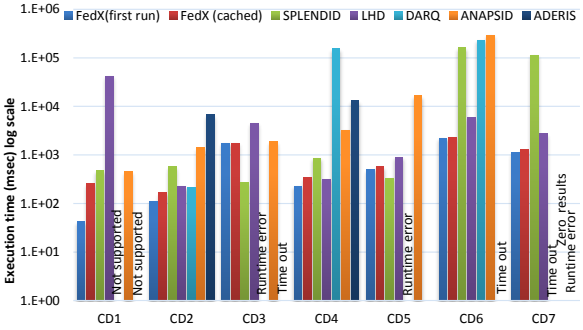
8. Conclusion

In this paper, we evaluated six SPARQL endpoint federation systems based on extended performance metrics and evaluation framework. We kept the main experimental metric (i.e. query execution time) unchanged and showed that the three other metrics (i.e. total triple pattern-wise selected sources, total number of SPARQL ASK request used during source selection, and source selection time), which did not receive much attention so far, can significantly affect the main metric. We also measured the effect of the data partitioning on these systems to test the effective parallel processing in each of the federation system. Overall, our results suggest that a combination of caching and ASK queries with accurate heuristics for source selection (as implemented in ANAPSID) has the potential to lead to a significant improvement of the overall runtime of federated SPARQL query processing systems.

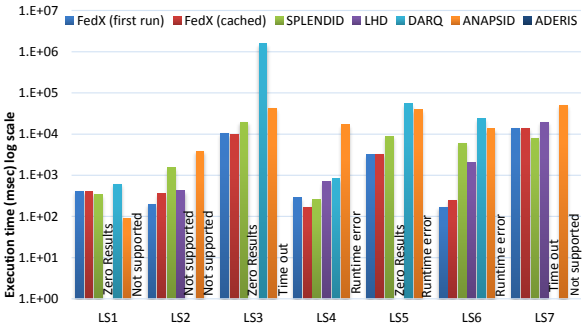
In future work, we will aim to get access to and evaluate the systems from our survey which do not provide a public implementation. We will also measure the effect of a range of various features (e.g., RAM size, SPARQL endpoint capabilities restrictions, vertical and hybrid partitioning, and duplicate detection) on the overall runtime of federated SPARQL engine. Furthermore, we will assess these systems on big data SPARQL query federation benchmark.



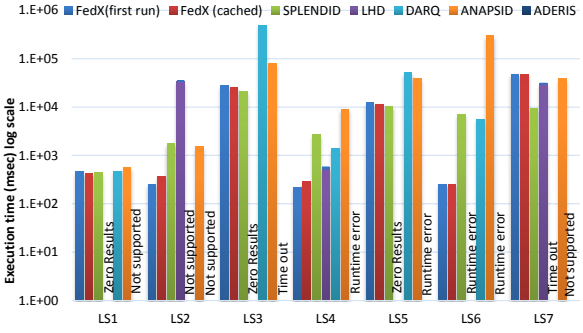
(a) FedBench: CD queries



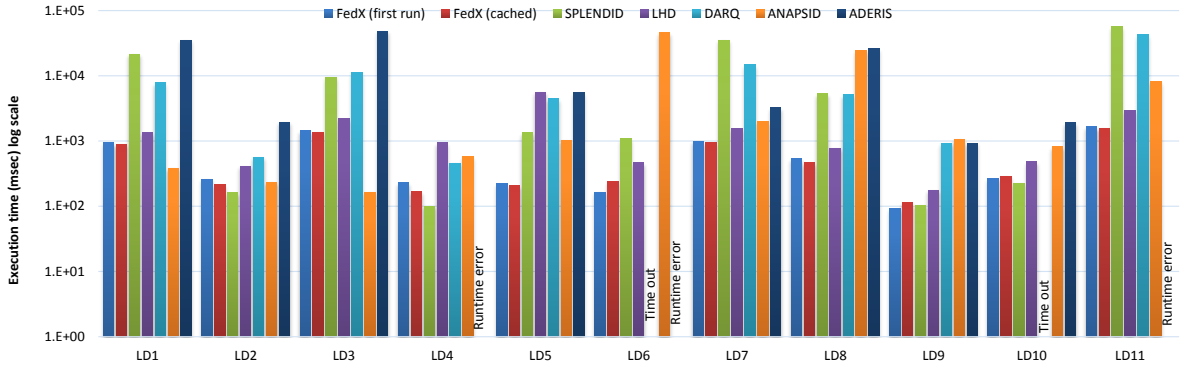
(b) SlicedBench: CD queries



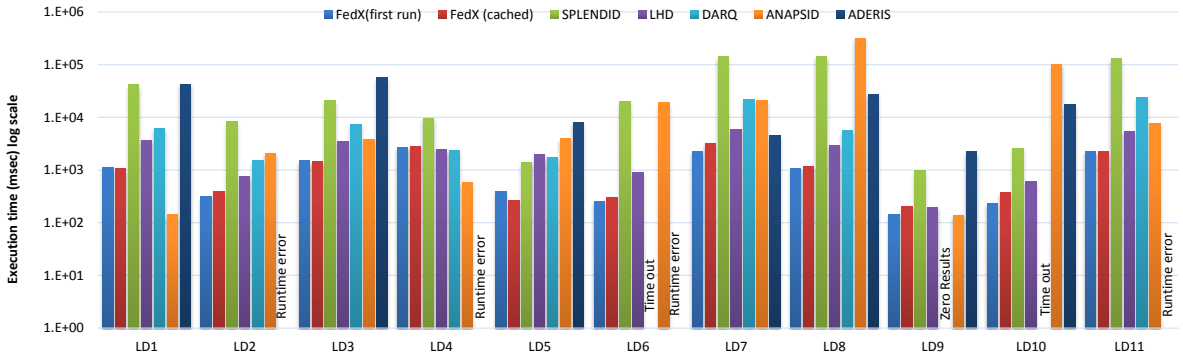
(c) FedBench: LS queries



(d) SlicedBench: LS queries

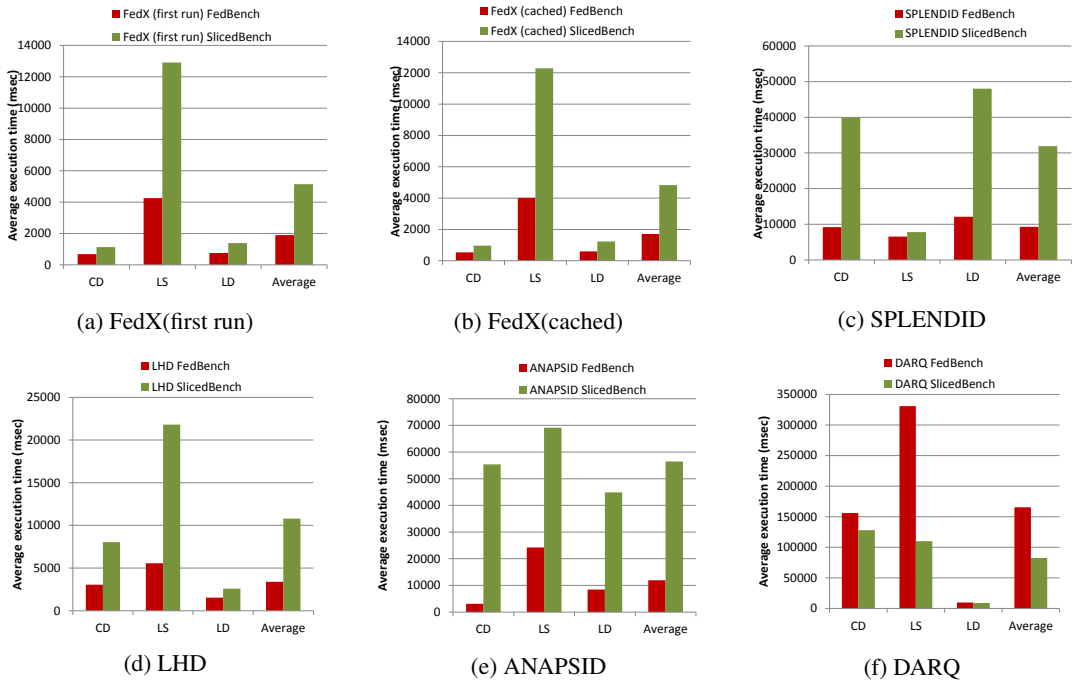


(e) FedBench: LD queries



(f) SlicedBench: LD queries

Fig. 7. Comparison of pure query execution time (without source selection time) for FedBench and SlicedBench



(a) FedX(first run) (b) FedX(cached) (c) SPLENDID

(d) LHD (e) ANAPSID (f) DARQ

Fig. 8. Effect of the data partitioning

9. Acknowledgments

This work was partially supported by the EU FP7 projects GeoKnow (GA: 318159) and BioASQ (GA: 318652) as well as the DFG project LinkingLOD.

References

[1] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: an adaptive query processing engine for sparql endpoints. In *ISWC*, 2011.

[2] Z. Akar, T. G. Halaç, E. E. Ekinci, and O. Dikenelli. Querying the web of interlinked datasets using void descriptions. In *LDOW at WWW*, 2012.

[3] F. Amorim. Join reordering and bushy plans. In *Simple Talk: A technical journal and community hub from Red Gate*, 2013.

[4] C. H. Andriy Nikolov, Andreas Schwarte. Fedsearch: efficiently combining structured queries and full-text search in a sparql federation. In *ISWC*. 2013.

[5] C. Basca and A. Bernstein. Avalanche: putting the spirit of the web back into semantic web querying. In *SSWS*, pages 64–79, November 2010.

[6] H. Betz, F. Gropengießer, K. Hose, and K.-U. Sattler. Learning from the history of distributed query processing - a heretic view on linked data management. In *COLD*, 2012.

[7] C. Bizer and A. Schultz. The berlin sparql benchmark. *IJSWIS*, 5(2):1–24, 2009.

[8] O. Görlitz and S. Staab. Splendid: Sparql endpoint federation exploiting void descriptions. In *COLD at ISWC*, 2011.

[9] Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158–182, 2005.

[10] O. Görlitz and S. Staab. Federated data management and query optimization for linked open data. In A. Vakali and L. Jain, editors, *New Directions in Web Data Management 1*, volume 331 of *Studies in Computational Intelligence*, pages 109–137. Springer Berlin Heidelberg, 2011.

[11] O. Hartig. An overview on execution strategies for linked data queries. *Datenbank-Spektrum*, pages 1–11, 2013.

[12] A. Hasnain, R. Fox, S. Decker, and H. F. Deus. Cataloguing and linking life sciences lod cloud. In *OEDW at EKAW*, 2012.

[13] A. Hasnain, M. R. Kamdar, P. Hasapis, D. Zeginis, C. N. Warren Jr, et al. Linked Biomedical Dataspace: Lessons Learned integrating Data for Drug Discovery. In *International Semantic Web Conference (In-Use Track), October 2014*, 2014.

[14] A. Hasnain, S. Sana E Zainab, M. R. Kamdar, Q. Mehmood, C. N. Warren Jr, et al. A Roadmap for navigating the Life Scinces Linked Open Data Cloud. In *International Semantic Technology (JIST2014) conference (To be appear), November 2014*, 2014.

[15] Y. E. Ioannidis and Y. C. Kang. Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization. In *ACM SIGMOD Record*, volume 20, pages 168–177. ACM, 1991.

[16] U. Juergen, H. Aidan, P. Axel, and D. Stefan. Link traversal querying for a diverse web of data. *Semantic Web Journal*, 2013.

[17] M. R. Kamdar, D. Zeginis, A. Hasnain, S. Decker, and H. F. Deus. ReVeaLD: A user-driven domain-specific interactive search platform for biomedical research. *Journal of Biomedical Informatics*, 47(0):112 – 130, 2014.

[18] Z. Kaoudi, M. Koubarakis, K. Kyzirakos, I. Miliaraki, M. Magiridou, and A. Papadakis-Pesaresi. Atlas: Storing, updating and querying rdf(s) data on top of dhts. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4), 2010.

[19] G. Ladwig and T. Tran. Linked data query processing strategies. In *ISWC*, pages 453–469. 2010.

[20] G. Ladwig and T. Tran. Sihjoin: Querying remote and local linked data. In *The Semantic Web: Research and Applications*, volume 6643, pages 139–153. 2011.

[21] S. Lynden, I. Kojima, A. Matono, and Y. Tanimura. Aderis: An adaptive query processor for joining federated sparql endpoints. In *OTM*, pages 808–817. 2011.

[22] G. Montoya, M.-E. Vidal, and M. Acosta. A heuristic-based approach for planning federated sparql queries. In *COLD*, 2012.

[23] G. Montoya, M.-E. Vidal, O. Corcho, E. Ruckhaus, and C. Buil-Aranda. Benchmarking federated sparql query engines: are existing testbeds enough? In *ISWC*, pages 313–324. 2012.

[24] M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. Dbpedia sparql benchmark - performance assessment with real queries on real data. In *International Semantic Web Conference*, pages 454–469, 2011.

[25] B. Quilitz and U. Leser. Querying distributed rdf data sources with sparql. In *ESWC*, pages 524–538, 2008.

[26] N. A. Rakhmawati, J. Umbrich, M. Karnstedt, A. Hasnain, and M. Hausenblas. Querying over federated sparql endpoints - a state of the art survey. volume abs/1306.1723, 2013.

[27] M. Saleem, R. Maulik, I. Aftab, S. Shanmukha, H. Deus, and A.-C. Ngonga Ngomo. Fostering serendipity through big linked data. In *SWC at ISWC2013*, 2013.

[28] M. Saleem, A.-C. Ngonga Ngomo, J. X. Parreira, H. F. Deus, and M. Hauswirth. Daw: Duplicate-aware federated query processing over the web of data. In *ISWC*, pages 561–576, 2013.

[29] M. Saleem, S. Shanmukha, A.-C. Ngonga Ngomo, J. S. Almeida, S. Decker, and H. F. Deus. Linked cancer genome atlas database. In *I-Semantics 2013*, 2013.

[30] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. Fedbench: a benchmark suite for federated semantic data query processing. In *ISWC*, pages 585–600. 2011.

[31] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. Spˆ 2bench: a sparql performance benchmark. In *ICDE*, pages 222–233, 2009.

[32] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *ISWC*, pages 601–616. 2011.

[33] A. Schwarte, P. Haase, M. Schmidt, K. Hose, and R. Schenkel. An experience report of large scale federations. *CoRR*, abs/1210.5403, 2012.

[34] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD*, pages 23–34, 1979.

[35] 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.

[36] X. Wang, T. Tiropanis, and H. C. Davis. Lhd: Optimising linked data query processing using parallelisation. In *LDOW at WWW*, 2013.