Undefined 1 (2009) 1–5
IOS Press

SPARQLES: Monitoring Public SPARQL Endpoints

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

Pierre-Yves Vandenbussche [a], Jürgen Umbrich [b], Aidan Hogan [c], and Carlos Buil-Aranda [d]
[a] *Fujitsu (Ireland) Limited, Swords, Co. Dublin, Ireland*
E-mail: pierre-yves.vandenbussche@ie.fujitsu.com
[b] *Vienna University of Economy and Business (WU), Austria*
E-mail: jurgen.umbrich@wu.ac.at
[c] *Centro de Investigación de la Web Semántica, Department of Computer Science, University of Chile, Chile*
E-mail: ahogan@dcc.uchile.cl
[d] *Centro de Investigación de la Web Semántica, Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile, Chile*
E-mail: cbuil@ing.puc.cl

Abstract. We describe SPARQLES: an online system that monitors the health of public SPARQL endpoints on the Web by probing them with custom-designed queries at regular intervals. We present the architecture of SPARQLES and the variety of analytics that it runs over public SPARQL endpoints, categorised by availability, discoverability, performance and interoperability. To motivate the system, we gives examples of some key questions about the health and maturation of public SPARQL endpoints that can be answered by the data it has collected in the past year(s). We also detail the interfaces that the system provides for human and software agents to learn more about the recent history and current state of an individual SPARQL endpoint or about overall trends concerning the maturity of all endpoints monitored by the system.

Keywords: SPARQL endpoints, Linked Data, Semantic Web, Web of Data

1. Introduction

Hundreds of Linked Datasets have been made publicly available in recent years. These datasets span a plethora of topics, varying from general-interest datasets like DBpedia [14][1] or GeoNames[2], to more niche datasets on topics like proteins[3] and Pokémon[4]. Despite their diverse subject matter, each dataset follows the Semantic Web standards [17,12] for describing its content, and the Linked Data principles [4] for making that content accessible on the Web. The goal is not only to enable clients to access these datasets in an automated and uniform way, but also to enable them to combine content from multiple locations in a similarly automated fashion.

To entice new consumers of their datasets, many publishers began hosting public SPARQL endpoints over their datasets such that clients can pose complex queries to the server as a single request and retrieve direct answers. Hundreds of public SPARQL endpoints have emerged on the Web in recent years, where Jentzsch et al. [13] estimated that 68% of the Linked Datasets in the 2011 LOD Cloud catalogue of-

[1] http://datahub.io/dataset/dbpedia
[2] http://datahub.io/dataset/geonames-semantic-web
[3] http://datahub.io/dataset/uniprot-databases
[4] http://datahub.io/dataset/pokepedia-fr

fered a link to a SPARQL endpoint. These endpoints index content with a variety of topics and sizes and accept arbitrary SPARQL queries over the Web.

However, applications using these endpoints have been slow to emerge. The convenience of SPARQL queries for clients translates into significant server-side costs maintaining such heavyweight query services. Leaving aside more general questions such as quality (e.g., correctness of data) or usability (e.g., what help is there for clients to create queries), a number of SPARQL-specific infrastructural challenges remain.

First, AVAILABILITY is a concern: SPARQL endpoints can become overloaded by numerous clients asking complex queries. Endpoints are often provided on a not-for-profit basis, where the resources available to host and maintain them may be limited and thus services may go offline temporarily or even permanently without warning. An application relying on a given endpoint would inherit these underlying availability issues; the situation can be even worse if an application relies on multiple endpoints.

Second, DISCOVERABILITY is an issue: finding an endpoint relevant to your needs can be quite difficult. Even if you know the types of classes or predicates you are looking for, or the keywords for the instances you are particularly interested in, or even simply the domain that you wish to explore, finding all of the available SPARQL endpoints that might be relevant is not a trivial task. Though de facto standards do exist for describing and advertising datasets that, if used, could help mitigate the discoverability issue, such descriptions are often not provided by endpoints [6].

Third, the PERFORMANCE of endpoints can be unpredictable. Evaluating a SPARQL 1.0 query is PSPACE-complete [16]; the analogous complexity for SPARQL 1.1 evaluation is at least as hard. Of course, these types of worst-case queries are likely to be quite rare [9], but even "PTIME queries" can require huge amounts of processing to satisfy over even moderate datasets. Thus the difficulty for a server of processing a single SPARQL query varies from trivial to infeasible, with everything in between. Even estimating *a priori* the cost of executing a query is hard. In addition, the performance of individual endpoints may differ for comparable workloads due to use of different SPARQL engines on servers of varying capabilities. Hence applications must cope with different levels of latency if working with public endpoints.

Finally, INTEROPERABILITY is a problem: once you have discovered some relevant SPARQL endpoints, you may find that the endpoints support different features of SPARQL to different degrees. In particular, some endpoints might support new features of SPARQL 1.1 like aggregates and sub-queries, whereas others might not.[5] This diversity in features supported means that you may not have a uniform query interface common to all endpoints against which you can program the logic of your application.

We previously performed a once-off analysis of 427 public SPARQL endpoints listed in the DataHub catalogue along the aforementioned dimensions of AVAILABILITY, DISCOVERABILITY, PERFORMANCE and INTEROPERABILITY, showing the extent to which these are issues in practice [6]. The analysis painted a mixed picture: while some endpoints performed well in the tests and thus might be candidates for application developers to rely on, many others performed poorly or had gone offline.

As such, we foresaw the need for a public system to track such aspects of endpoints over time. We thus initiated work on the SPARQLES (SPARQL Endpoint Status) system, available at `http://sparqles.okfn.org`.[6] The system has been online for the past year, during which time we have been making various refinements based on feedback collected from the community. Meanwhile we have collected a year's worth of initial experimental data that can answer novel questions about public endpoints: are they getting faster, are more SPARQL 1.1 features being supported, are they becoming better described, etc.

This paper thus extends upon previous works [6,18] and describes the current SPARQLES system itself in detail: how it is constructed, what sorts of analytics it performs, what queries it issues, what data it collects, what kinds of conclusions can be drawn, what visualisations are provided and what APIs are provided.

We begin with the high-level architecture.

2. SPARQLES Architecture

The SPARQLES system is designed to observe a set of public SPARQL endpoints over time. Currently SPARQLES is tracking all the endpoints listed in the

[5] Though not covered herein, one such issue is the non-standard support for keyword search, which varies from engine to engine.

[6] SPARQLES is also a predecessor of an older system that tracked only availability [18]: `http://labs.mondeca.com/sparqlEndpointsStatus/`; l.a. 2015/01/30.

DataHub catalogue[7] found using the DataHub APIs; we thus align the inclusion criteria of SPARQLES with that of DataHub. SPARQLES performs a fixed set of analytics against each listed endpoint at fixed intervals, stores the historical results and allows them to be accessed or visualised through online interfaces.

The high-level architecture for observing the selected endpoints is depicted in Figure 1, where we show the offline and online parts of the system. The offline parts are responsible for collecting information about the endpoints. The online parts are responsible for presenting the results to the clients of the system. The main components are as follows:

Analytics **(*offline*):** responsible for performing analysis over endpoints at regularly defined intervals, thus producing the raw observational data.

Storage **(*both*):** offers persistence over the results of the offline Analytics component and enables online querying and aggregation.

A.P.I. **(*online*):** offers software agents a RESTful application programming interface through which to query key data about endpoints.

U.I. **(*online*):** offers human agents a user interface with a mix of aggregate visualisations and per-endpoint visualisations.

In Section 3, we describe the offline phase, and in particular, the types of analytics we run; to help motivate these analytics, we present some research questions that can be answered from the data collected by the system thus far. Thereafter, in Section 4, we describe the storage and online parts of the system, including the types of interfaces that we provide for the public to interact with the collected data.

3. Analytics

We now provide details of the offline phase: we describe the analytics performed by the system and give some insights into the types of conclusions that can be drawn from the data collected thus far.

3.1. Availability

From our previous experiments, we have found that many endpoints have significant periods of downtimes [6]. Some downtimes may be temporary, caused by network failures, sporadic high server loads, engine crashes, and so forth. Other downtimes appear permanent, indicating that an endpoint is probably being discontinued. Anticipating downtimes or distinguishing reliable endpoints from unreliable ones can be crucial for many clients. Hence SPARQLES closely monitors the historical availability of endpoints.

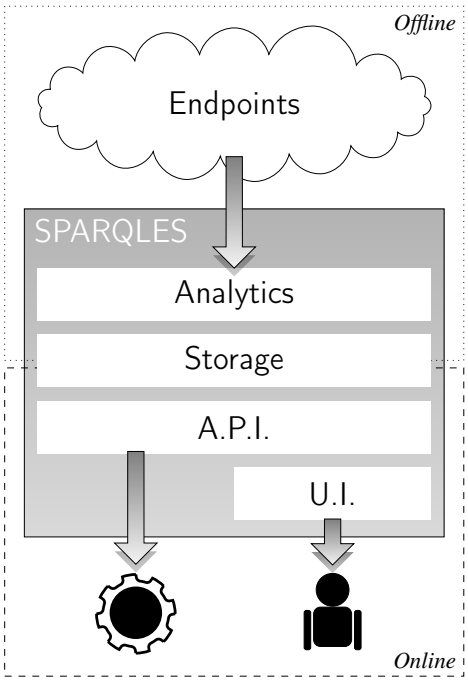

Fig. 1. High-level System Architecture

Availability analytics SPARQLES assesses the availability of each SPARQL endpoint on an hourly basis. We define an endpoint as available if it can respond to a simple SPARQL query with some compliant response.

To check availability, the system issues a generic `ASK` query as follows:

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

Responding to this query should be trivial (is the index empty or not?). As soon as a valid response (positive or negative) is received, the system considers the request successful and concludes that the endpoint is available. However, some SPARQL endpoints cannot handle this `ASK` query. For such endpoints, we try a second query using the `SELECT` operation as follows:

```
SELECT ?s WHERE { ?s ?p ?o . } LIMIT 1
```

[7] `http://datahub.io/`; l.a. 2015/01/30.

We deem any endpoint responding to either query with any valid SPARQL response as suitable.

Availability results are then aggregated per endpoint into a success rate for fixed time intervals to generate uptime estimates at different levels of granularity; e.g., the last day, the last week, the last month. Clients can access this information through various interfaces in the SPARQLES system, which will be described later in Section 4.2.

Data Analysis We have been monitoring the hourly availability of endpoints since February 2011.[8] As of the end of November 2014, when we began to collate our results, about ten million test queries had been executed. To motivate the collection of this dataset by the SPARQLES system, we present two research questions that the corpus can help to answer.

A1: *How has the number of (un)reliable public endpoints evolved down through the years?*

To help initially answer this question, Figure 2 shows the evolving availability for endpoints spanning a period of February 2011 to November 2014, grouped by monthly availability percentage (ticks are bimonthly). As a global trend, we see an overall increase in the number of endpoints year on year (downward dips refer to endpoints that are explicitly deleted from the DataHub catalogue).[9] We also see a growing number of endpoints falling into one of two extremes ($[0,5]$ and $]99,100]$), with a more stable number of relatively few endpoints in the aggregate $]5,99]$ interval: the majority of endpoints tend to either die off completely or to have availability in excess of 99%.

To a certain extent, endpoints going offline is to be expected: many such endpoints were simply experimental in nature. Having characterised the extent of this phenomenon, henceforth we filter dead endpoints from consideration and focus on 344 live endpoints that were available at least once during November 2014, which leads to our second question:

[8] In fact, this data collection pre-dates the modern SPARQLES system currently described, where the first three years of data were gathered by a predecessor system: `http://labs.mondeca.com/sparqlEndpointsStatus/`; l.a. 2015/01/31.

[9] Interestingly, the large growth in the number of endpoints between July and August 2011 corresponded with the announcement of an upcoming update of the LOD Cloud, resulting in increased submissions to the CKAN/DataHub catalogue: `http://lists.w3.org/Archives/Public/public-lod/2011Jul/0059.html`; l.a. 2015/01/29.

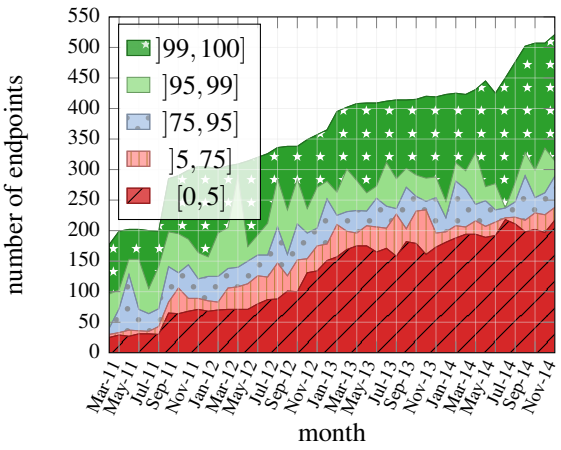

Fig. 2. Evolution of number of endpoints falling into different availability rates between February 2011 and November 2014

A2: *Considering only live endpoints, how has availability evolved?*

Figure 3 removes dead endpoints from consideration and normalises the y-axis as a ratio of overall endpoints. The plot spans from November 2013 to November 2014, in order to correspond with the timespan found in later experiments (prior to November 2013 and the release of the modern SPARQLES system, we only ran availability experiments). We see that of the remaining endpoints, about 5% still tend to have extended periods of downtime before coming back online. Conversely, we see that generally between 40–60% of these endpoints have availability in excess of 99% (i.e., "two nines" availability).

Note that in future sections, we likewise focus on results for these 344 "live endpoints".

Limitations One of the main limitations of the availability experiments are problems that are local to the SPARQLES engine: the local server may experience some downtimes or local network issues. In general however, when errors known to be local are omitted and when hourly results are aggregated into larger time intervals, such as weeks or months, such local effects should be smoothed out.

3.2. Discoverability

For a client, finding a SPARQL endpoint that contains content relevant for their needs [15,6] and the features that they require [6] can be challenging. The goal of the discoverability analytics is to determine the degree to which endpoints offer descriptions of them-

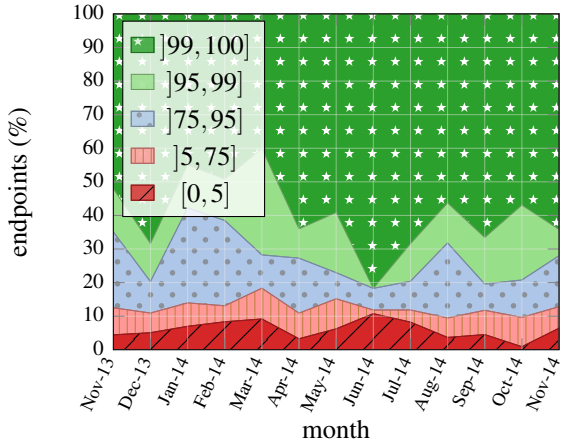

Fig. 3. Evolution of ratio of endpoints per availability rate between November 2013 and November 2014

selves and their contents using (de facto) standards: to what extent an endpoint offers sufficient descriptive meta-data such that it can be discovered by a client. In particular, the SPARQLES system checks if a client can automatically find, for a given endpoint:

1. Meta-data about the indexed datasets in the form of VoID descriptions [2].
2. Meta-data about the capabilities and graphs indexed by the endpoint in the form of SPARQL 1.1 Service Descriptions [21].
3. The type of engine powering the endpoint, sometimes mentioned in the HTTP header.

VoID Analytics The Vocabulary of Interlinked Datasets (VoID) [2] has become the de facto standard for describing RDF datasets (in RDF). The vocabulary allows for specifying, e.g., an OpenSearch description, the number of triples a dataset contains, the number of unique subjects, a list of properties and classes used, number of triples associated with each property (used as predicate), number of instances of a given class, number of triples used to describe all instances of a given class, predicates used to describe class instances, and so forth. Likewise, the description of the dataset is often enriched using external vocabulary, such as for licensing information.

If VoID descriptions were widely available for SPARQL endpoints, a client could leverage them to discover endpoints with potentially relevant content. SPARQLES thus tracks for which endpoints such descriptions are available in a discoverable location.

Given a SPARQL endpoint URL as input, the system checks in two locations for a VoID file. The first location is that constructed from the Well-Known URI pattern `http://{domain}/.well-known/void` recommended for use with VoID, where `{domain}` is replaced with the fully-qualified domain name (FQDN) extracted from the endpoint URL.[10] Secondly, the system checks if the endpoint has indexed its own description using the following query, where `%%ep` is replaced with the URL of the endpoint:

```
PREFIX void: <http://rdfs.org/ns/void#>
SELECT DISTINCT ?ds
WHERE { ?ds a void:Dataset ;
  void:sparqlEndpoint %%ep . }
```

SD Analytics Endpoint capabilities – such as supported SPARQL version, query and update features, I/O formats, custom functions, and/or entailment regimes – can be described in RDF using the SPARQL 1.1 Service Description (SD) vocabulary, which became a W3C Recommendation in March 2013 [21]. Such descriptions, if made widely available, could help a client find public endpoints that support the features it needs (e.g., find SPARQL 1.1 endpoints).

The service description for an endpoint is retrieved by simply dereferencing the endpoint URI itself [21]. As such, the SPARQLES system performs a HTTP GET request for an endpoint URI, follows redirects and uses content negotiation to request RDF formats (viz. RDF/XML, N-Triples, Turtle or RDFa).

Server Name Analytics A variety of options are now available for SPARQL engines, including Virtuoso [7], Sesame [5], 4store [10], etc. However, performance and compliance across different vendors can vary quite dramatically. Knowing which engine – or even which version of an engine – powers a given SPARQL endpoint may be useful for (expert) clients to know which version of a query to send. For example, in previous works we found that certain analogous strategies for processing joins in a federated setting worked well for certain SPARQL engines but performed poorly or even outright failed for others [3].

Unfortunately, neither VoID nor the SD vocabulary provide terms for specifying an engine or version number to a client. Hints are available, such as scanning the frontpage or an error page for mention of a fixed list of engines. However, when dereferencing the endpoint URL, the type of engine and the version number is often (though not always) provided in

[10] `http://vocab.deri.ie/void/autodiscovery`; l.a. 2015-01-27.

the `Server` field of the HTTP header. Although not always provided – perhaps since it may require low-level server configurations – this is the cleanest method we have found to currently establish which implementation powers an endpoint without requiring hard-coded, engine-specific heuristics.

Data Analysis To help motivate these analytics, we discuss two questions that can be answered using the discoverability data collected by SPARQLES:

D1: *Are endpoints adding VoID and SD descriptions?*

To answer this question, we analyse the meta-data available for endpoints in November 2013 and January 2015 to see if anything has changed over the 14 months.[11] We focus on 234 live endpoints that were listed in both November 2013 and January 2015, for which, we look in the various aforementioned locations for VoID and SD descriptions. We also look in the VoID Well-Known location for SD meta-data and also at the endpoint URL for VoID meta-data, since both could be published together as a combined description.

Table 1 presents the results. In general, we see that few descriptions were added for legacy endpoints. In fact, we see a decline in the number of endpoints offering VoID meta-data, particularly as part of the index (the majority of these endpoints were on the `bio2rdf.org` domain). We see that overall, the ratio of the 234 endpoints with descriptions is quite low, and that few return both flavours of meta-data.[12]

Although few legacy endpoints added descriptions of themselves, we found that of the 110 new endpoints coming online in those 14 months, 41 offered SD meta-data and 15 offered VoID meta-data.

D2: *Do endpoints change their underlying engine?*

Taking the same endpoints on the same dates as before, we looked at the `Server` entry in the HTTP headers and compared the breakdown of engine counts between November 2013 and January 2015 to see if there was any growth in the popularity of a given engine.

[11] Unfortunately, during paper writing, we found a problem in the configuration of the discoverability analytics left behind after a server migration, meaning that data for 2014 could not be included. As such, we are limited to a once-off comparison in the results between November 2013 and January 2015.

[12] In one case where a VoID Well-Known file contained only SD and not VoID, the cause was a redirect; see, e.g., `http://sparql.openmobilenetwork.org/.well-known/void` (l.a. 2015-01-29); which redirects to an endpoint.

Table 1
Changes in available meta-data from 2013 to 2015

Meta-data	Location	2013	2015
VoID	Well-Known	51	53
	Dereferencing	8	8
	Direct Query	70	61
	Any	**89**	**78**
SD	Well-Known	6	9
	Dereferencing	47	47
	Any	**53**	**54**
Both	Well-Known	6	8
	Dereferencing	1	2
	Any	**21**	**13**

Table 2
Changes in reported `Server` values in HTTP headers

Server	2013	2015
`Apache`	71	74
`Virtuoso`	75	72
—	25	36
`Apache-Coyote`	23	15
`Fuseki`	11	13
...	...	...
`RDF::Endpoint`	0	1

The results are given in Table 2, where generic HTTP-server entries are greyed out. We see two new types of SPARQL engines being reported, but only with a single entry: Sesame and RDF::Endpoint.

Looking through changes for individual endpoints, 35 endpoints changed the server-name value in the HTTP response header: 5 systems provide a more specific server name (e.g., from Apache to Fuseki), 2 systems changed the underlying engine (from one RDF-specific engine to another), 19 systems removed the server entry, 4 systems added a server name, and 5 systems provide a more general name than before (e.g., from Virtuoso to Apache).

The general conclusion is thus that endpoints rarely change implementation and that – based on incomplete `Server` headers – the relative popularity of different engines is quite stable, with Virtuoso the most popular, followed by Fuseki.

Limitations SPARQLES only checks for the existence of meta-data, but does not attempt to validate the meta-data itself, nor does it try to measure the completeness of descriptions. Additionally, VoID descrip-

tions or engine information may be extracted from other locations not checked by SPARQLES: however, as per a client, we believe it is important to offer such information in generic, discoverable locations.

3.3. Performance

SPARQLES runs a set of performance-related analytics that aims to compare the runtimes of different public endpoints for comparable queries from a client's perspective (i.e., including HTTP overhead). Since we cannot control or know in detail about the content of endpoints, for the purpose of comparability, we must rely on generic queries that would execute in a similar manner independent of the exact content indexed by the endpoint. We test three fundamental aspects of a query engine: lookups, streaming and joins.

Lookup Analytics The goal is to measure the time taken to perform an atomic lookup (according to different triple patterns). The query template is as follows:

```
ASK {<x> ?p ?o}
```

In this case, `<x>` is replaced with an arbitrary IRI that is not expected to exist in the remote data (a lookup still needs to be performed to ensure this).

Since in the above example the *s*ubject is set, we call it an $\texttt{ASK}_s$ query. We also run $\texttt{ASK}_p$, $\texttt{ASK}_o$, $\texttt{ASK}_{sp}$, $\texttt{ASK}_{so}$, $\texttt{ASK}_{po}$, $\texttt{ASK}_{spo}$ versions of the query.

Given that an atomic lookup should be fast to execute, we expect that the performance of such queries would be dominated by the HTTP overhead.

Streaming Analytics We measure the time taken for an endpoint to stream a large result-set that should be trivial to compute. The query is as follows:

```
SELECT * {?s ?p ?o} LIMIT 100001
```

Here we ask to stream 100,001 results. Since we have found that public endpoints may limit maximum result sizes to a "round number" – say 100,000 – we ask for one hundred thousand *and one* results to detect such a case. We also send queries for limits with 50,000, 25,000, 12,500, 6,250 and 3,125 results.

Join Analytics We use the following three queries to measure a generic notion of join performance:

```
SELECT DISTINCT ?s ?q
WHERE {?s ?p ?o OPTIONAL {?s ?q <x>}} LIMIT 1000
```

```
SELECT DISTINCT ?s ?q
WHERE {?s ?p ?o OPTIONAL {<x> ?q ?s}} LIMIT 1000
```

```
SELECT DISTINCT ?o ?q
WHERE {?s ?p ?o OPTIONAL {<x> ?q ?o}} LIMIT 1000
```

These queries are designed – insofar as possible – to be comparable across endpoints no matter what content is indexed. In these queries, `<x>` is an arbitrary URI not expected to appear in the data. For example, the first query requests that 1,000 unique subjects be joined with a pattern that generates no answers: this join must still be executed to check that `?q` is indeed unbound. The result will return 1,000 distinct subject–unbound pairs. While the first query looks at *s*–*s* joins, the second performs an *s*–*o* join and the third an *o*–*o* join. (These three join-types were the most common found in analyses of real-world logs [9].)

Data Analysis The SPARQLES system schedules performance analytics once a day for public endpoints. In the following, we give two examples of questions about the performance of public SPARQL endpoints that can be answered using the data collected thus far by SPARQLES spanning November 2013 to November 2014. In theory, we should have around 30 datapoints for every month. However, due to downtimes of the SPARQLES system in this period, we have only between 8–15 results for November 2013, December 2013 and March 2014. We consider these enough datapoints to aggregate and thus include these months.

The first question we pose is as follows:

P1: *Is the performance of SPARQL endpoints improving with time?*

One may expect that as underlying SPARQL implementations and internet infrastructure improve, the performance of endpoints would benefit over time.

Figures 4–6 provide the evolution of performance for `ASK`, `JOIN` and `LIMIT` queries respectively. These plots are generated as follows:

- Only live endpoints are included
- We only include response times for `ASK` queries that return a correct answer (`false`), and for `JOIN` and `LIMIT` queries where the number of results is within 99–101% of the expected total.
- For a fixed query (e.g., $\texttt{ASK}_s$), we take a monthly median value for each endpoint.

– We filter endpoints without valid median values for all months. The number of endpoints considered is showed in the legend on the figures.
– For each month, we choose the median from all the endpoints as the final value shown.

We use median rather than mean values since the performance data is often positively skewed [6]: median values are more robust to outliers (i.e., a few query executions that take thousands of times longer to answer than the typical case would skew the mean).

From the figures, we can observe that endpoints struggle more for $\texttt{ASK}_o$, $\texttt{ASK}_s$ and $\texttt{ASK}_{spo}$ than other types of `ASK` queries, and that they struggle for $\texttt{JOIN}_{oo}$ queries more than other joins involving a subject. However, caution is required in comparing the different series. In Figures 6, for example, we see that `LIMIT 25000` is slower than `LIMIT 50000`, which is due to the fact that a different set of endpoints is considered: while 105 endpoints consistently return enough results to be counted in the `LIMIT 25000` series, 25 of these must be discounted from the `LIMIT 50000` series, possibly due to indexing fewer than 50,000 triples or implementing a results threshold [6]. Hence different endpoints are being compared.

Rather than comparing across series, our focus is on **P1**: does the performance of endpoints improve over time? In general, a dramatic improvement in performance is not evident from the data. Although some of the `ASK` queries in Figure 4 show some variance from month-to-month, the overall trend is stable. The `JOIN` queries in Figure 5 do show some steady but minor improvement over the course of the experiments. For the `LIMIT` queries, there is a dramatic improvement from the first month; however, these queries generate the largest result sizes, and we think the slow initial queries may be due to some network effects rather than a major simultaneous improvement in the performance of the engines powering the endpoints.[13]

The second question we look at is as follows:

P2: *To what extent are the response times for a given query and a given endpoint consistent over time?*

In other words, how often do query response times for a given endpoint and query fall out of the scope

[13] It is tempting to suggest that the poor performance of the `LIMIT` queries in May 2014, as visible in Figure 6, may have been due in some part to the May 12th deadline of ISWC 2014. But this hypothesis is admittedly not backed up by the other query types.

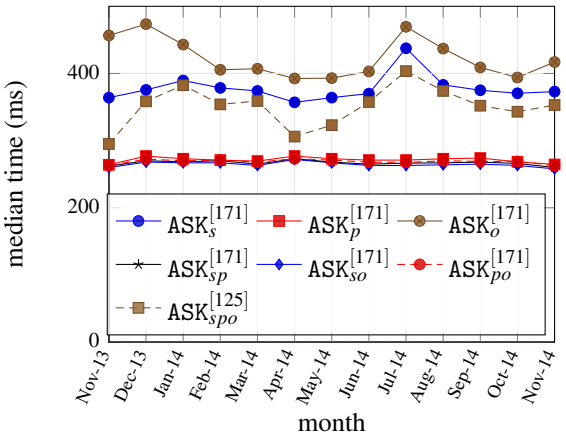

Fig. 4. Evolution of ASK query performance

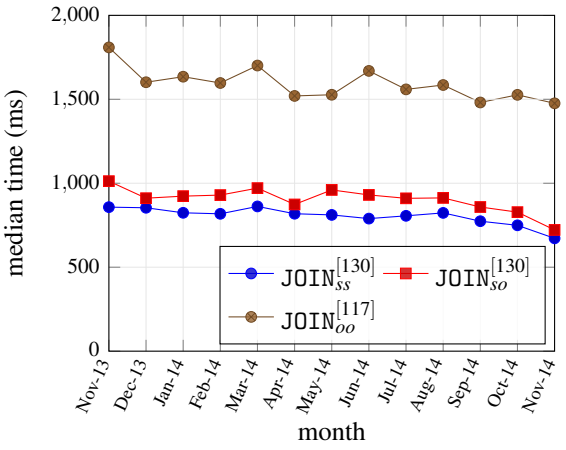

Fig. 5. Evolution of JOIN query performance

of "typical performance"? We take the median runtime of a given query for a given endpoint as our indicator of "typical performance", and then look at how many queries take 1.1× the typical median runtime, or 2× or 0.5×, and so on. Figure 7 presents such an analysis:

– For a given query and a given endpoint, we compute the median query response time over the length of the experiment.
– We compute the ratio of all response times against that median for that query and endpoint.
– We group all of these ratios into experiment type (`ASK`|`JOIN`|`LIMIT`) and sort them.
– We then plot the cumulative number of queries with at least that ratio.

For example, we see in Figure 7 that ∼50% of queries were faster than median performance (and thus

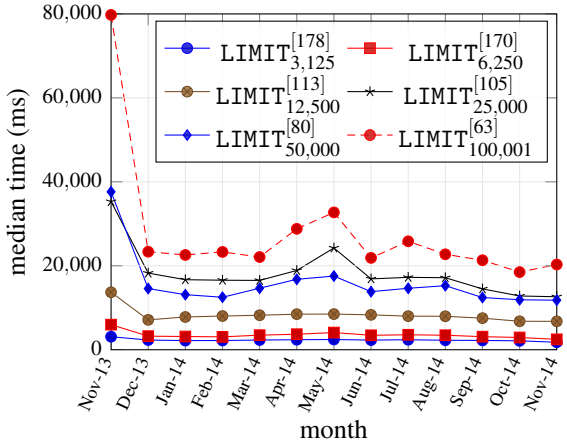

Fig. 6. Evolution of LIMIT query performance

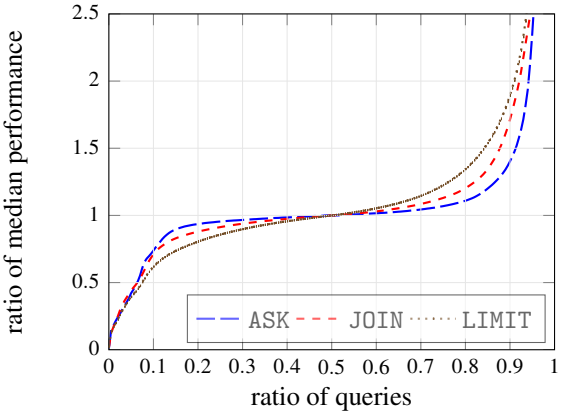

Fig. 7. Stability of query performance

∼50% were slower), as must hold by definition. But we also see that, e.g., in the ASK series, 10% of queries were answered in less than 0.75× the median time; conversely, we see that 90% of ASK queries were answered within 1.75× the median runtime. Thus, if the endpoints and experiments gave very stable results, this would display as a line close to $y = 1$, whereas we see in reality, e.g., that 1-in-5 JOIN queries will take 25% longer than expected, and so forth. In fact, in the interest of readability, the graph is trimmed to show a maximum of 2.5× the median performance: some outlier queries had runtimes hundreds or thousands of times longer than the median performance for that endpoint (the highest was 4,286× for an ASK query).

In summary, we can see that about 70% of the queries (0.1–0.8 in the x-axis of Figure 7) run within about 0.75–1.25× the median performance.

Limitations The performance results do not indicate *why* specific queries are slow: is it due to the engine, the HTTP overhead, the content indexed? In general, we try to make the query load balanced irrespective of the content and our goal is to measure the costs from the perspective of a client who is concerned about the "bottom line" of response times.

Perhaps one of the main limitations of the performance results is again the issue of local effects. For example, slow runtimes may be due to a busy network on the SPARQLES end, or, conversely, endpoints on geographically close servers may be given an advantage. Likewise in early months of the experiments, we experienced local downtimes; however, we resolved the particular issues that caused these downtimes, resulting in more stable analytics in later months.

3.4. Interoperability

If available, the Service Description of an endpoint should describe the query features and the version of SPARQL that an endpoint supports. However, we have seen that SD meta-data are often unavailable and, in any case, an endpoint may claim to support features that it does not, or may claim support for SPARQL 1.1 while only supporting a subset of new features.

SPARQLES thus offers analytics for interoperability, whose goal is to verify which SPARQL features – i.e. specific operators, solution modifiers, etc. – are supported, gathering data about what SPARQL features are available for the users of various endpoints.

Along these lines, SPARQLES takes a subset of queries from the W3C Data Access Working Group test-cases – designed to test all features from both versions of the standard – and issues them on a weekly basis to SPARQL endpoints. We consider the test as passed if a valid SPARQL response is returned. Since we cannot control the content of endpoints, we cannot verify that the returned response is actually correct; hence we may overestimate compliance with the standard. We expect that if an endpoint does not support a feature, an exception will be thrown (e.g., a parse exception). However, since an endpoint may time-out on a given query, we may also underestimate compliance where the feature may be supported but the endpoint cannot answer the query instance provided.

SPARQL 1.0 Analytics First, the SPARQLES system tests the endpoints for the core SPARQL 1.0 query features that it supports. We issue endpoints a subset of the Data Access Working Group test-cases for

SPARQL 1.0,[14] omitting syntax tests and focusing on core functionalities.[15] This test-set checks a range of aspects of the SPARQL 1.0 standard including query types including `SELECT`, `CONSTRUCT` and `ASK` (we omit `DESCRIBE` since it is an optional feature); filter features, such as `REGEX`, IRI and blank node checks, etc.; support for datatypes, such as numerics, strings and booleans; support for graph selection features, including `FROM (NAMED)` and `GRAPH`; and the solution modifiers, `ORDER BY`, `LIMIT` and `OFFSET` (`DESC|ASC`), as well as `DISTINCT` and `REDUCED` modifiers.

SPARQL 1.1 Analytics SPARQLES also performs tests on SPARQL 1.1 features using a test suite taken from the W3C SPARQL Working Group.[16] We omit testing for SPARQL 1.1 Update since we (hopefully) will not have write privileges for public endpoints. We do not test entailment since, without knowledge of the content, we cannot verify if results are entailed or not.

We first test support for aggregates, where expressions such as average, maximum, minimum, sum and count can be applied over groups of solutions (possibly using an explicit `GROUP BY` clause). We then test support for sub-queries in combination with other features. Next we test support for property-paths, binding of individual variables, and support for binding tuples of variables (`VALUES`). We also check support for filter features that check for the existence of some data (`MINUS`, `EXISTS`), and some new operator expressions (`STRSTARTS` and `STRCONTAINS` for strings; `ABS` for numerics). Finally, the last three queries test a miscellany of features including `NOT IN` used to check a variable binding against a list of filtered values, an abbreviated version of `CONSTRUCT` queries whereby the `WHERE` clause can be omitted, and support for the SPARQL `SERVICE` keyword.

Data Analysis SPARQLES collects data about interoperability on a weekly basis. To motivate this collection, we look at the following question:

I1: *Is support for SPARQL 1.1 features increasing with time?*

SPARQL 1.1 became a W3C Recommendation in March 2013 [11]; hence we are interested to look at how support for new features developed through 2014.

Since there are 18 queries in total issued specifically to test features new to SPARQL 1.1, in order to allow more succinct presentation, we group these queries according to their purpose, as follows:

FILTER Tests the following filter features and other operators: `IN`, `EXISTS`, `NOT EXISTS`, `MINUS`, `ABS`, `CONTAINS`, `ABS`, `STRSTARTS`.

AGGREGATES Tests the following aggregate features: `AVG`, `MIN` `SUM`.

MISCELLANY Tests various features: `BIND`, `MINUS`, property paths, `SERVICE`, sub-queries, and `VALUES`.

In Figure 8 we present the ratio of endpoints passing these groups of SPARQL 1.1 features. The time interval for the queries was again November 2013 to November 2014, inclusive. The results were constructed as follows:

- We consider a query as passed by an endpoint in a month if we received at least one valid SPARQL response to that query in that month.
- We consider a group as passed if one of the queries passes.

These conditions are perhaps quite "generous" to endpoints, but we assume that an endpoint will not remove features, nor is it likely, e.g., to support one aggregate but not another.

The results show that the level of support for each group of queries is quite similar across all months, and that support for the different groups is quite similar. This may indicate that when an endpoint deploys a SPARQL 1.1 engine, it tends to support a broad range of new SPARQL features rather than incrementally adding features. During August 2014, there was a visible increase in the amount of endpoints passing the test (50% of endpoints returned results); we are uncertain what caused this spike. Aside from that month, we see that about 40% of endpoints have broad support for SPARQL 1.1, and that during 2014, there was a subtle increase in SPARQL 1.1 adoption, moving from about 40% to 45% of endpoints not returning errors.[17]

Limitations As aforementioned, the main limitation of these experiments is that we classify an endpoint as implementing a specific SPARQL 1.1 feature if that endpoint returns any valid response without throwing an exception. If an endpoint times out, we will classify it as not implementing that feature, and conversely, if

[14] `http://www.w3.org/2001/sw/DataAccess/tests/r2`; l.a. 2015/01/30.

[15] Queries available at `https://github.com/pyvandenbussche/sparqles`.

[16] `http://www.w3.org/2009/sparql/docs/tests/data-sparql11/`; l.a. 2015/01/31.

[17] All the exceptions encountered are reported by SPARQLES for individual endpoints/queries; details are available on the web site.

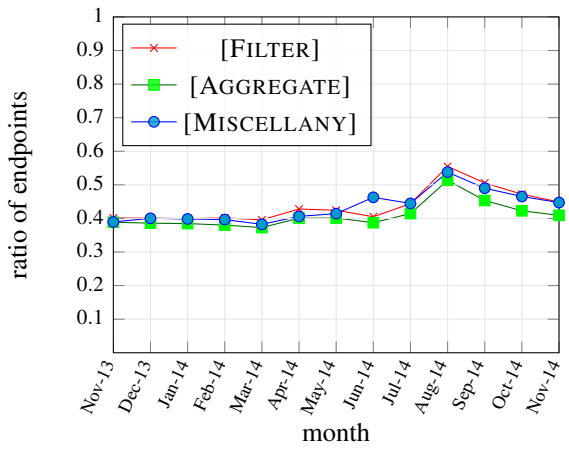

Fig. 8. Evolution of support for SPARQL 1.1 feature groups

it returns an incorrect solution, we will count it as supporting that feature.[18]

Another limitation was brought to our attention by the community: the W3C test-case queries used by SPARQLES, in their unmodified form, were potentially very expensive for servers to run.[19] Hence, in the past month, we have refactored these queries to include URIs that will not appear in the data. The results presented herein are based on the original queries.

4. Storage & Interfaces

We now describe the storage used by SPARQLES to manage the data gathered by the experiments previously described, as well as the public interfaces through which software agents and users can interact with the data collected.

The SPARQLES system – incorporating both code and data – is published under a Creative Commons 4.0 license (CC BY), with code available from `https://github.com/pyvandenbussche/sparqles` and data available from interfaces that will be described later in the section.

[18] Strictly speaking, a query timing out is not compliant with SPARQL 1.1; however, in spirit, we are more interested about whether a feature is supported in general and not about if a specific query instance runs or not.

[19] In fact, a member of the British Library told us in personal communication that the hosting costs for their SPARQL services jumped from around £1,000 to around £3,000 due to expensive queries being issued, some of which were coming from SPARQLES.

4.1. Storage

Queries and other requests (e.g., discoverability checks) are handled using Apache Jena (2.12.2).[20] The analytics presented – as well as checking DataHub for updates to the endpoint list – are scheduled to run at regular intervals using cron jobs. As tests are performed, the results and metrics collected are serialised using the Apache AVRO (1.7.5) library and sent to a MongoDB instance for storage.

The MongoDB instance maintains 11 different collections that, loosely speaking, represent different materialised views over the data collected:

- 4 collections store the "raw" version of the data collected for the four analytical dimensions;
- 1 collection maintains the current list of endpoints registered in the DataHub;
- 6 collections correspond to aggregated views of the raw data as required by the User Interface.

The aggregate views are recomputed at regular intervals using cron jobs: these views return the data required by the U.I. in a single lookup and thus avoid running aggregations while the user waits.

Over the last year, we observed a storage index growth of 120 MB per month during data collection.

4.2. Application Programming Interfaces

SPARQLES provides a set of RESTful APIs that allow remote access to the data collection.[21] The basic design guide for these APIs is that they should allow applications to get access to the same information humans have access to via the User Interfaces. Unlike the high-level results used to motivate the data collection herein, the SPARQLES system offers low-level information for individual endpoints. More precisely, per Figure 9, various APIs offer access to endpoint-specific as well as global analytical information.

Endpoint-specific functions The LIST and AUTOCOMPLETE functions allow to locate SPARQL endpoints of interest. The INFO function provides comprehensive, up-to-date information about one endpoint including its availability (for the last test, last day, last week, last month and overall); its latest performance details with the suspected result size threshold based on the `LIMIT` queries mentioned in Section 3.3; its

[20] `https://jena.apache.org`; l.a. 2015/01/30.

[21] `http://sparqles.okfn.org/api`

compliance with each of the 24 SPARQL 1.0 features and the 18 SPARQL 1.1 features as part of the interoperability dimension; and its discoverability (server name and (non-)existence of VoID and SD meta-data).

High-level analytical functions The analytical functions provide a way to access an overview for each of the four dimensions with respect to all SPARQL endpoints monitored in SPARQLES. The AVAILABILITY function returns the list of endpoints with their associated uptime for the last test, the last day and last week. The DISCOVERABILITY function provides information of server name, and the accessibility of VoID and SD meta-data for all endpoints. Similarly the INTEROPERABILITY and PERFORMANCE functions allow an application to access respectively: the compliance to SPARQL 1.0 and SPARQL 1.1 recommendations; and the mean performance for `ASK` and `JOIN` queries.

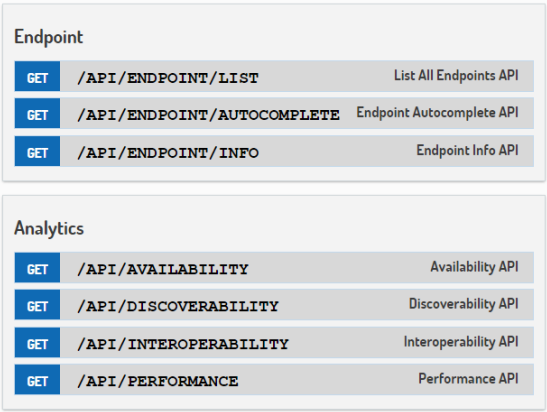

Fig. 9. List of APIs to access SPARQLES data

4.3. User Interface

The SPARQLES user interface – available at `http://sparqles.okfn.org` – offers an entry point for human users interested in the experiments. The interface is implemented using various Javascript libraries, including, e.g., Node.js[22] and nvd3[23] for rendering interactive visualisations.

The homepage offers "at-a-glance" aggregated views of the four dimensions computed for all endpoints. From there, a user has a number of possible navigation steps, as illustrated in Figure 10. The user can navigate to more detailed information about one of the dimensions wherein a summary of the results for all endpoints are provided in a list view. Otherwise, a user can either use an auto-complete search function on the endpoint URL, or click on an endpoint URL in a list view, to find detailed, up-to-date information about all four dimensions for a given endpoint. From the homepage, the user can also find links for data dumps and for API documentation.

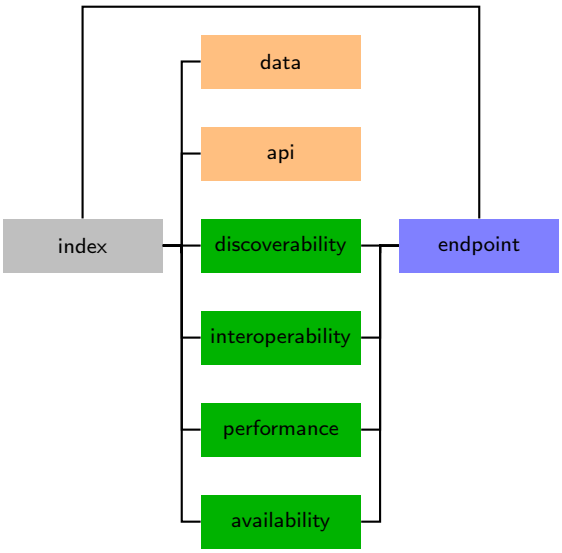

Fig. 10. SPARQL Website sitemap.

5. Discussion

We now discuss high-level issues regarding the impact, limitations and sustainability of SPARQLES.

5.1. Impact

One of the main goals of the system is to disseminate timely information about the health of individual endpoints. The online SPARQLES site receives on average about 500 unique visitors per month, 48% of which have visited before in the year previous.

Another indirect goal of the system is to encourage endpoints to follow best practices: we would hope that by tracking such metrics about endpoints, maintainers might be made aware of shortcomings with the SPARQL services they offer and rectify these accordingly. Though from personal communications with some endpoint maintainers, we know that there have been anecdotal instances of this, the results in this pa-

[22] `http://nodejs.org/`; l.a. 2015/30/01.
[23] `http://nvd3.org/`; l.a. 2015/30/01.

per tend to suggest the effect has not been so dramatic: for example, we have not seen a huge increase in VoID or SD meta-data since the system came online.

Perhaps the most important impact of this work thus far has been to formally acknowledge the kinks in the current public SPARQL infrastructure, which has helped motivate new lines of research. We can, for example, point to works proposing Linked Data Fragments – an alternative method for accessing Linked Dataset aiming at high availability by reducing server costs – which draws heavily upon the availability statistics from our original analysis to justify why alternatives to SPARQL are needed [20,19]. We can also point to works like SHEPHERD [1], which uses the statistics about query performance to generate more efficient query plans for public SPARQL endpoints, or to works by Netahu et al. [8] on profiling datasets for the purposes of enabling better discoverability, or indeed to our own work on taking the weaknesses of endpoints into account when creating federated query plans [3].

5.2. Limitations

For each of the analytics presented in Section 3, we discussed a variety of specific limitations, referring, e.g., to the difficulty in distinguishing local problems from remote problems. There are also a couple of global limitations of the system worth mentioning.

First, SPARQLES is subject to Goodhart's law:

> *When a measure becomes a target, it ceases to be a good measure.*

An over-eager endpoint maintainer could, for example, detect and artificially respond to SPARQLES queries so as to improve how the endpoint is "rated" by the system. In general, we know of no such example of this happening and trust that it will not happen.

Second, as a more pragmatic issue, since we first put the system online in November 2013, we have had various local reliability issues, where data were not collected for certain weeks, where data were lost due to server migration, and where the site itself was offline. During this period, we have been resolving various issues as they occur such that, although there are still some known issues, we now believe that the system is reaching maturity. Likewise, we have received a lot of feedback from the community, which has been invaluable for improving the service in the past year.

5.3. Sustainability

SPARQLES is (generously) hosted by the Open Knowledge Foundation, who have pledged continued support for the system into the future.

One indirect but important aspect of sustainability is the load that SPARQLES puts on the public SPARQL infrastructure. For example, we discussed before about how the original versions of the interoperability queries were causing a heavy load for a number of SPARQL services. To mitigate this, we run more expensive tasks less frequently: while simple availability tests are done hourly, performance analytics are run daily and interoperability tests are run weekly. Likewise we have recently revised the interoperability queries to make them less costly.

As the system has been maturing, we have started to consider adding some new features as requested by the community. One of the most popularly request features is to have data collected by the SPARQLES tool made available as Linked Data. Though we are (perhaps ironically) reluctant to make a SPARQL endpoint available, as a starting point, we are looking into creating Linked Data URIs for individual endpoints that dereference to SPARQLES statistics about them. Other requested features included offering an email notification system to contact endpoint administrators when their system was not available, or offering badges for endpoints with high availability, and so forth.

6. Conclusion

In this paper, we have presented the SPARQL Endpoint Status (SPARQLES) system for keeping track of the health and maturity of public SPARQL endpoints. We presented the high-level architecture, which consists of an offline component for running tests over endpoints, and an online component for providing visualisations and APIs for the collected results. We presented four dimensions of analytics that the system runs over public endpoints, which we helped motivate by presenting some key questions about SPARQL endpoints that the resulting data could be used to answer. Thereafter, we presented some of the details of how SPARQLES is implemented and the interfaces provided for human and automated agents to interact with the underlying data collection. Finally, we discussed some aspects relating to the high-level impact, limitations and sustainability of the tool.

In general, we believe that the SPARQLES system provides the community with a unique window into public SPARQL endpoints, offering a view that has helped to inspire some novel research directions. The system has shed light not only on some cobwebs and cracks in the SPARQL infrastructure, but also on the cream of the crop: those SPARQL endpoints that are highly-available, readily-discoverable, highly-performant and highly-interoperable.

Acknowledgements This work was supported by Fujitsu Laboratories Limited, by CONICYT/FONDECYT Project no. 3130617, by CONICYT/FONDECYT Project no. 11140900, and by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004. We would like to broadly thank all of the members of the Linked Data community who have offered their feedback and suggestions about SPARQLES through the project page, mailing lists, and personal communications. We would also like to warmly thank the Open Knowledge Foundation for agreeing to host the project.

References

[1] M. Acosta, M. Vidal, F. Flöck, S. Castillo, C. B. Aranda, and A. Harth. SHEPHERD: A shipping-based query processor to enhance SPARQL endpoint performance. In *ISWC 2014 Posters & Demons*, pages 453–456, 2014.

[2] K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. Describing linked datasets. In *LDOW*. CEUR (Vol. 538), 2009.

[3] C. B. Aranda, A. Polleres, and J. Umbrich. Strategies for executing federated queries in SPARQL1.1. In *ISWC 2014*, pages 390–405, 2014.

[4] T. Berners-Lee. Linked Data. Design issues for the World Wide Web, World Wide Web Consortium, 2006. `http://www.w3.org/DesignIssues/LinkedData.html`.

[5] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In *ISWC*, pages 54–68, 2002.

[6] C. Buil-Aranda, A. Hogan, J. Umbrich, and P. Vandenbussche. SPARQL web-querying infrastructure: Ready for action? In *ISWC*, pages 277–293, 2013.

[7] O. Erling and I. Mikhailov. RDF support in the virtuoso dbms. In *Networked Knowledge – Networked Media*. Springer, 2009.

[8] B. Fetahu, S. Dietze, B. Pereira Nunes, M. Antonio Casanova, D. Taibi, and W. Nejdl. A scalable approach for efficiently generating structured dataset topic profiles. In *The Semantic Web: Trends and Challenges*, volume 8465, pages 519–534, 2014.

[9] M. A. Gallego, J. D. Fernández, M. A. Martínez-Prieto, and P. D. L. Fuente. An empirical study of real-world SPARQL queries. In *USEWOD Workshop*, 2012.

[10] S. Harris, N. Lamb, and N. Shadbolt. 4store: The design and implementation of a clustered RDF store. In *SSWS*, 2009.

[11] S. Harris and A. Seaborne. SPARQL 1.1 query language. W3C Recommendation, March 2013.

[12] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph. OWL 2 Web Ontology Language Primer. W3C Recommendation, Oct. 2009. http://www.w3.org/TR/owl2-primer/.

[13] A. Jentzsch, R. Cyganiak, and C. Bizer. State of the LOD cloud. Public Web-page, September 2011.

[14] 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*, 6, 2015. (to appear).

[15] H. Paulheim and S. Hertling. Discoverability of SPARQL endpoints in Linked Open Data. In *Proceedings of the ISWC 2013 Posters & Demonstrations Track, Sydney, Australia, October 23, 2013*, pages 245–248, 2013.

[16] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.

[17] G. Schreiber and Y. Raimond. RDF 1.1 Primer. W3C Working Group Note, June 2014. `http://www.w3.org/TR/rdf11-primer/`.

[18] P. Vandenbussche, C. B. Aranda, A. Hogan, and J. Umbrich. Monitoring SPARQL endpoint status. In *ISWC Posters & Demos*, pages 81–84, 2013.

[19] R. Verborgh, O. Hartig, B. D. Meester, G. Haesendonck, L. D. Vocht, M. V. Sande, R. Cyganiak, P. Colpaert, E. Mannens, and R. V. d Walle. Low-cost queryable linked data through triple pattern fragments. In *ISWC 2014*, pages 13–16, 2014.

[20] R. Verborgh, M. V. Sande, P. Colpaert, S. Coppens, E. Mannens, and R. V. de Walle. Web-Scale Querying through Linked Data Fragments. In *Workshop on Linked Data on the Web*, 2014.

[21] G. T. Williams. SPARQL 1.1 Service Description. W3C Recommendation, March 2013.