Undefined n (2016) 1–14
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], Luca Matteis [c], Aidan Hogan [d] & Carlos Buil-Aranda [e]
[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] *Department of Computer Science, Sapienza University of Rome, Italy*
E-mail: matteis@di.uniroma1.it
[d] *Centro de Investigación de la Web Semántica, Department of Computer Science, University of Chile, Chile*
E-mail: ahogan@dcc.uchile.cl
[e] *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. 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. We likewise present some details of the performance and usage of the system thus far.

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

1. Introduction

Thousands of Linked Datasets have been made publicly available in recent years.[1] These datasets span a plethora of topics, varying from general-interest datasets like DBpedia [20][2] or GeoNames[3], to more niche datasets on topics like proteins[4] or Pokémon[5]. Each dataset follows the Semantic Web standards [32,19] for describing its content, and the Linked Data principles [7] for making that content accessible on the Web. The goal is to enable clients to access these diverse datasets in an automated and uniform way, and also to combine content from multiple locations in a similarly automated fashion.

To entice new consumers, 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. Hun-

[1] At the time of writing, LODstats [11] reports 9,960 datasets: `http://stats.lod2.eu/` (l.a.: November 29, 2015). The most recent "State of the LOD Cloud" report found 1,014 datasets.
[2] `http://datahub.io/dataset/dbpedia`
[3] `http://datahub.io/dataset/geonames-semantic-web`
[4] `http://datahub.io/dataset/uniprot-databases`
[5] `http://datahub.io/dataset/pokepedia-fr`

0000-0000/16/$00.00 © 2016 – IOS Press and the authors. All rights reserved

dreds of public SPARQL endpoints have thus emerged on the Web in recent years [9]. These endpoints index content with a variety of topics and sizes and (in theory at least) accept arbitrary SPARQL queries from remote clients 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, which translate into a variety of technical problems on the level of the SPARQL infrastructure itself [9].

With respect to what that SPARQL infrastructure consists of, the recent SPARQL 1.1 standard issued recommendations relating to the following:

Query The SPARQL 1.1 Query Language recommendation [18] extends the original SPARQL Query Language [27] with features such as property paths, sub-queries, aggregates, etc. The related SPARQL 1.1 Federated Query recommendation [26] specifies how a SPARQL engine can invoke a remote endpoint at runtime.

Protocol The SPARQL 1.1 Protocol recommendation [12] specifies how clients should interact with a SPARQL endpoint over HTTP, including how `GET`/`POST` requests should be structured, what sorts of responses should be returned, etc. Three output result formats have also been recommended, extending the XML format introduced in the original standard with options for returning data in JSON or CSV/TSV.

Description The SPARQL 1.1 Service Description recommendation [36] provides a vocabulary with which the capabilities and configuration of a SPARQL endpoint can be described in RDF such that, for example, clients can discover endpoints with the features they need.

Update The SPARQL 1.1 Update specification [15] describes a language for inserting, deleting and updating the data present in a SPARQL engine.

Entailment The SPARQL 1.1 Entailment Regimes recommendation [16] describes how ontological entailments can be included when computing the answers for a SPARQL query.

With respect to public SPARQL endpoints, the latter two aspects of the SPARQL infrastructure are *currently* of lesser interest: updates to data are unlikely to be enabled on a public query service and we do not yet know of any public SPARQL endpoint supporting entailment regimes. Thus, for clients of current public endpoints, the former three aspects – query, protocol, and description – appear to be of most relevance.

With respect to these three infrastructural aspects, in previous work [9] we performed an empirical investigation of the maturity of public SPARQL endpoints from the perspective of the client, who we argue needs endpoints that are: (i) highly-available through the SPARQL protocol, thus allowing queries to be answered reliably at any time; (ii) described using standard vocabularies in well-known locations, thus allowing for the (automatic) discovery of relevant endpoints over the Web; (iii) capable of answering queries in acceptable time, thus enabling their use in real-time applications; (iv) compliant with respect to supporting the query features of SPARQL (1.1), thus enabling them to be interrogated alongside other endpoints in a uniform manner. If an endpoint has high availability, is well-described, supports all features of SPARQL 1.1, and returns query answers quickly, we consider it to meet all of the basic infrastructural requirements a client would have. However, these goals fall on a continuum rather than being binary or discrete: for example, endpoints may support a majority of features of SPARQL 1.1, or may only be able to answer certain queries within a given expected response time. Thus the question is to *what extent* are endpoints mature. We thus defined four general dimensions for assessing the maturity of public SPARQL endpoints, as follows.

First, we looked at AVAILABILITY: *the ratio of time for which a given endpoint is responsive through the SPARQL protocol*, or alternatively, *the probability of a SPARQL endpoint being able to successfully respond to a (simple valid) SPARQL query at a given point in time*. In our study, we found that many of the endpoints listed on the DataHub had issues with availability: we found that on average, over 27 months, 29.3% of the endpoints were online 0–5% of the time, and that 32.2% were available more than 95% of the time [9]. 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. Likewise, executing SPARQL queries can be expensive for a server, which may reach its capacity and be unable to respond to further requests. 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, we looked at DISCOVERABILITY: *the degree to which an endpoint provides descriptions of*

its content, configuration and functionality in well-known locations using well-known vocabularies, such that clients can (automatically) discover that endpoint based on criteria such as the classes and properties its content pertains to, the amount of data it contains, the query features it supports, etc. For the endpoints in our survey, we checked the availability of associated Service Description (SD) [36] and Vocabulary of Interlinked Datasets (VoID) [2] descriptors in well-known locations: we found VoID meta-data for approximately one-third of the endpoints (mostly in external catalogues, such as DataHub), and SD meta-data for approximately one-tenth of the endpoints [9]. Without these meta-data, clients may struggle to find endpoints with content and features relevant for their needs.

Third, we looked at PERFORMANCE: *the amount of time taken for an endpoint to answer a query over HTTP using standard SPARQL protocol methods*. Running generic forms of queries over the endpoints surveyed, we found that for comparable query loads, the slowest 10% of the endpoints took around 45% of the overall experiment time, and that the simplest form of query – a basic `ASK` query – took around 300 ms in the median case; as a side result, we also found that for performance reasons, many endpoints limited the maximum number of results returnable, with the most common threshold being 10,000 [9]. Evaluating a SPARQL 1.0 query is PSPACE-complete [25]; 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 [14], but even "PTIME queries" can require huge amounts of processing to satisfy over even moderate datasets. In our experiments, even sticking to "modest" queries with a bounded number of joins, intermediate results and final results, we still encountered slow response times and partial results for some endpoints, which may make them unsuitable for use in real-time applications.

Finally, we looked at INTEROPERABILITY: *how compliant the endpoint is with respect to the features of the SPARQL 1.1 query language*. In particular, an endpoint that does not support some query features of SPARQL 1.1 may not be interoperable with other endpoints or applications that expect these features to be supported. In our survey, we found that support for SPARQL 1.1 features – which was recently standardised at the time – was patchy, with for example 7% supporting federated queries (with the `SERVICE` keyword). This diversity in features supported means that a client may not have a uniform query interface common to all endpoints against which they can program the logic of their application(s).

Given the mixed results of our initial experiments, we foresaw the need for an online system to track such aspects of public endpoints over time, and to help clients assess for themselves the maturity of individual endpoints based on the empirical tests. Along these lines, we initiated work on the SPARQL Endpoint Status (SPARQLES) system, which is currently available at `http://sparqles.ai.wu.ac.at/`.[6] The SPARQLES system has been online since October 2013 (two years at the time of writing), during which time we have made various refinements based on community feedback, and have made the system more reliable and less expensive for the public endpoints we monitor. This paper extends upon previous works [9,33] and describes the current SPARQLES system itself in detail: how it is constructed, what sorts of tests it performs, what queries it issues, what data it collects, what kinds of conclusions can be drawn, what interfaces and visualisations are provided, etc.

In Section 2, we first discuss works relating to studies of public endpoints and monitoring Web services. In Section 3, we introduce the high-level SPARQLES architecture. In Section 4 we describe in more detail the analytics that SPARQLES runs over public endpoints and in Section 5 we describe the interfaces that we provide for agents to interact with the data collected. In Section 6 we present evaluation of the system including runtimes of analytics, growth in storage overheads, and A.P.I. performance. We later discuss the impact, limitations, and sustainability of SPARQLES in Section 7 before concluding with Section 8.

The SPARQLES system – both code and data – is published under a Creative Commons 4.0 license (Ⓒⓘ), with code available from `https://github.com/pyvandenbussche/sparqles` and data available through interfaces described in Section 5.

2. Related Work

A number of works and systems have dealt with issues relating to public SPARQL endpoints. The DataHub[7] catalogue lists hundreds of Linked Datasets,

[6] SPARQLES is also a predecessor of an older system that tracked only availability [33]: `http://labs.mondeca.com/sparqlEndpointsStatus/`; l.a. 2015/01/30.

[7] `http://datahub.io/`, (l.a.: November 29, 2015).

many of which link to a SPARQL endpoint; it is the resulting list of endpoints that we monitor. Lorey [21] proposed a number of metrics for determining the performance of SPARQL endpoints, such as latency, throughput, random access time and join execution; experiments were performed in controlled settings rather than over public endpoints on the Web. Paulheim & Hertling [23] tried to find relevant SPARQL endpoints for a random sample of ten thousand IRIs; using VoID descriptions and the DataHub catalogue, they were successful in about 15% of cases. Mehdi et al. [22] proposed best-effort methods to find public endpoints relevant to a list of domain-specific keywords by querying endpoints for RDF literals generated from the terms. With respect to how public SPARQL endpoints are being used, a number of works have also performed analyses of the query logs of prominent endpoints [14,29], which were made available by the USEWOD initiative [6] and later by LSQ [31]. While all of these works have helped to build a more detailed picture of the current state-of-the-art with respect to public SPARQL endpoints, none provide an online system like SPARQLES, nor does any one work look at the range of analytics we provide.

More generally, since public SPARQL endpoints can be considered as Web Services, our work also relates to the topic of monitoring Web Services, and in particular, the notion of Quality of Service (QoS). One of the seminal works in this area was by Ran [28], who proposed an influential list of twenty-three QoS dimensions for Web services in four categories: runtime (R), transaction support (T), configuration (C), and security (S). In this context, our work touches upon the following dimensions:

Performance (R) The response time, latency and throughput of the service.
Availability (R) The probability of a system being operational at any given point in time.
Robustness/Flexibility (R) The degree to which a service can cope with diverse inputs.
Exception Handling (R) The gracefulness with which errors are handled and explained.
Supported Standard (C) The degree of compliance of the service with respect to some standards.
Completeness (C) The ratio of advertised features that are found to work in practice.

Some of the other QoS dimensions defined by Ran [28] are either not currently relevant for public SPARQL endpoints – such as authentication, authorization, etc. – or cannot be easily tested in our setting – for example, the accuracy of results. On the other hand, we also consider the discoverability of endpoints, which refers to how well the service describes itself, which was not explicitly mentioned by Ran [28].

3. SPARQLES Architecture

The SPARQLES system is designed to observe a set of public SPARQL endpoints over time. Currently SPARQLES is tracking all of the endpoints listed in the DataHub catalogue[8] found using the DataHub APIs; we thus align the inclusion criteria of SPARQLES with that of the DataHub. SPARQLES performs a fixed set of analytics against each listed endpoint at fixed intervals, stores the historical results and allows these results to be accessed 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 regular intervals, thus producing the raw observational data.

- This component is implemented with custom Java code that uses Jena as a query client to interact with endpoints over the SPARQL protocol. Analytics are scheduled using `cron` jobs.

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

- For this, we use MongoDB, which stores both low-level data – such as the responses of endpoints to individual queries – as well as higher-level, aggregated data – such as the number of queries that succeeded in the past month.

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

- Agents may access the A.P.I. through simple HTTP `GET` calls, which return JSON-formatted data from the storage back-end.

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

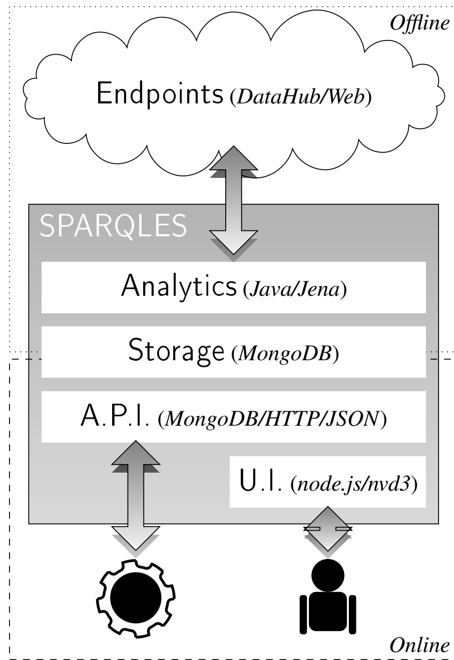


Fig. 1. High-level System Architecture

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

- The search and user interactions provided by the U.I. are built on top of the data delivered by the A.P.I. The user interface is implemented using various Javascript libraries, including Node.js[9] and nvd3[10] for rendering interactive visualisations.

We describe these components in more detail in the following sections. In Section 4, we focus on the offline phase, and in particular, the types of analytics we run. Thereafter, in Section 5, 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 by the SPARQLES system.

4. Analytics

We now provide details of the offline phase, and in particular, the analytics performed by the system.

The following analytics are implemented with custom Java code that uses Apache Jena (2.12.2) to coordinate making requests to and collecting responses from SPARQL endpoints.[11] These analytics are scheduled to run at regular intervals using `cron` jobs. SPARQLES is hosted in the Vienna University of Economy and Business (WU), served by a 1 Gigabit network.

4.1. Availability

From our previous experiments, we have found that many endpoints have significant periods of downtime [9]. 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 has probably been 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.

Availability analytics We define an endpoint as available if it can respond to a simple SPARQL query with some compliant response through the SPARQL protocol. To avoid unnecessary load on remote servers, we send queries that should, in general, be as simple as possible to compute responses for.

To check availability, the system first 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
```

Again, this query should be cheap to compute: return any triple from the index (if any). We deem any endpoint responding to either query with any valid SPARQL response as available at that time.

Schedule We run availability tests once an hour, which allows us to monitor, e.g., the uptimes at different times of the day, including hours of peak Web-usage (performance will be discussed later). Availability results can then be aggregated per endpoint into a success rate for fixed time intervals, e.g., to compute availability over the past day, week, month, etc.

[9] `http://nodejs.org/`; l.a. 2015/09/09.
[10] `http://nvd3.org/`; l.a. 2015/09/09.
[11] `https://jena.apache.org`; l.a. 2015/01/30.

Limitations The local SPARQLES server may experience some downtimes or local network issues that may lead to remote endpoints being falsely reported as unavailable. 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.

4.2. Discoverability

For a client, finding a SPARQL endpoint that contains content relevant for their needs [24,9] and the features that they require [9] can be challenging. The goal of the discoverability analytics is to determine the degree to which endpoints offer descriptions of themselves and their contents using (de facto) standards: to what extent an endpoint offers descriptions – in well-known locations using well-known vocabularies – of (i) its content and (ii) the features it supports. The SPARQLES system thus checks if a client can automatically find, for a given endpoint:

1. An SD description of its configuration [36].
2. A VoID description of its content [2].

The type of engine (Fuseki, Virtuoso, etc.) powering a SPARQL endpoint can also be an important information for a client; for example, some engines support non-standard keyword search functions that a client may be interested in. We thus also look for:

3. The type of engine powering the endpoint, sometimes mentioned in the HTTP header [9].

SD Analytics Endpoint capabilities – such as the version of SPARQL supported, 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 [36]. 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 IRI itself [36]. As such, the SPARQLES system performs a HTTP GET request for an endpoint IRI, follows redirects and uses content negotiation to request RDF formats (viz. RDF/XML, N-Triples, Turtle or RDFa).

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., the number of triples a dataset contains, the number of unique subjects, a list of properties and classes used, the number of triples with a given property as predicate, the number of instances of a given class, the number of triples used to describe instances of a given class, and so forth. If VoID descriptions were widely available for SPARQL endpoints, a client could leverage them to discover endpoints with potentially relevant content.

There are a number of best-practices regarding how VoID should be published; SPARQLES looks in three locations. First, the system looks in the content gotten by dereferencing the endpoint URL (i.e., the same document as the SD description). Second, the system checks the location denoted by the Well-Known IRI 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.[12] Third, SPARQLES uses the following query to detect if the endpoint indexes its own VoID description, where `%%ep` is replaced with the URL of the endpoint in question:

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

Server Name Analytics A variety of options are now available for SPARQL engines, including Virtuoso [10], Sesame [8], 4store [17], 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 the `Server` field of the HTTP header. Although not al-

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

ways provided – perhaps since it may require low-level server configuration – this is the cleanest method we have found to currently establish which implementation powers an endpoint without requiring hard-coded, engine-specific heuristics.

Frequency When compared with availability, we do not expect discoverability to be so dynamic: once descriptions are published, they are likely to stay published (and as discussed later, we do not check that the descriptions are up-to-date). For this reason, we run discoverability analytics once a week; we have received no complaints from endpoint maintainers about the remote expense of these analytics.

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 descriptions or engine information may be extracted from locations or with vocabularies not checked by SPARQLES: however, to help clients, we believe it is important to offer such information using well-recognised vocabularies in discoverable locations.

4.3. 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}
```

Here `<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). To mitigate caching effects, we generate a fresh IRI each time.

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

Given that an atomic lookup should be fast to execute, we consider this query as estimating the latency of querying the endpoint, which would most likely be dominated by the HTTP network 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. Since the endpoint should be able to stream results contiguously from its index, we consider this query as giving an estimate of the maximum throughput of the service.

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 fresh IRI 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 [14,31].)

Frequency Like availability, we expect performance to vary for different times of the day, different days of the week, etc. For this reason, like availability, we would like to have frequent experiments. However, unlike availability, the queries required to test performance are not so trivial for endpoints to compute (we present more details on this later). For this reason, we opted to run performance experiments once a day; at this level of frequency, we have received no complaints from endpoint maintainers.

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.

Performance results may also be affected by local issues. For example, slow runtimes may be due to a busy network on the SPARQLES end (e.g., if other analytics happen to run simultaneously); to help mitigate this issue, in the system’s interfaces, we display the median value of the last ten performance runs. Other factors may be more difficult to control for; e.g., endpoints on servers that are geographically closer to the SPARQLES host may be given an advantage. Still, the performance results should serve as a useful guide.

4.4. 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,[13] omitting syntax tests and focusing on core functionalities.[14] This test-set checks a range of aspects of the SPARQL 1.0 standard including query types `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.[15]

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, which invokes a federated request from the remote endpoint being tested to a local endpoint we have set up on the SPARQLES server.

Frequency Much like discoverability, we do not expect interoperability to be a highly dynamic property of an endpoint; for example, we suppose that once an endpoint adds support for SPARQL 1.1 features, it will continue to support these features until it is discontinued. For this reason, we schedule interoperability analytics to run once a week. During the first year of operation, we began to receive complaints that the queries we were using – based on the W3C test-cases – were

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

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

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

causing a high load on remote servers.[16] We considered lowering the frequency of these analytics but instead decided to make the queries less demanding by refactoring them to include fresh IRIs in such a way that it should be efficient for the server to compute that the result is empty; as usual, we then simply monitor for exceptions. With these new queries, we have received no complaints thus far.

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 it returns an incorrect solution, we will count it as supporting that feature.[17] Our recent refactoring of the test-case queries has helped to reduce false negatives due to time-outs. Another limitation is that the `SERVICE` call relies on our local endpoint being accessible; to mitigate problems, we designed the federated call to be cheap and simple and to restart our local endpoint just before calling these analytics.

Another limitation is that we do not test some features of the SPARQL 1.1 standard, such as SPARQL 1.1 Update – since we presume we should not have write privileges for public endpoints – or SPARQL 1.1 Entailment Regimes – since we do not know of any public SPARQL endpoints with this feature.

5. Storage & Interfaces

We now describe how SPARQLES manages the experimental data gathered during these analyses and the public interfaces through which software agents and users can interact with these data.

5.1. Storage

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.

5.2. Application Programming Interfaces

SPARQLES provides seven RESTful APIs that allow remote access to the data collection. These APIs are designed to provide clients with both (i) information relating to specific endpoints, as well as (ii) information about all endpoints relating to a specific type of analytical experiment. Likewise we split the APIs into two groups, as follows.

The first group contains *endpoint-specific APIs*, which helps to find a specific endpoint, or returns detailed results for a specific endpoint:

LIST takes no input and returns a list of all endpoints in SPARQLES: the URL of the endpoint, as well as the name and URL of the dataset they are associated with in the DataHub catalogue.

AUTOCOMPLETE takes as input a string, such as "`dbpedia`", and returns all endpoints in SPARQLES whose URL, dataset label, or dataset URL contains the input as a sub-string.

INFO takes as input the URL of a specific endpoint, and returns all data for that endpoint, including dataset URL and label; availability over the past day, week, month, and overall; performance results for cold/warm runs of ask and join queries and suspected threshold size; interoperability information regarding support for SPARQL and SPARQL 1.1 query features; and discoverability information regarding locations (if any) where VoiD and SD descriptions could be found, and the server-name extracted from the HTTP header.

The second group contains *analytical APIs*, which return aggregate results for all endpoints on a given analytical dimension:

AVAILABILITY returns uptimes for the last test, day, week, month – and overall – for all endpoints.

[16] See, e.g., `https://github.com/pyvandenbussche/sparqles/issues/23`.

[17] 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.

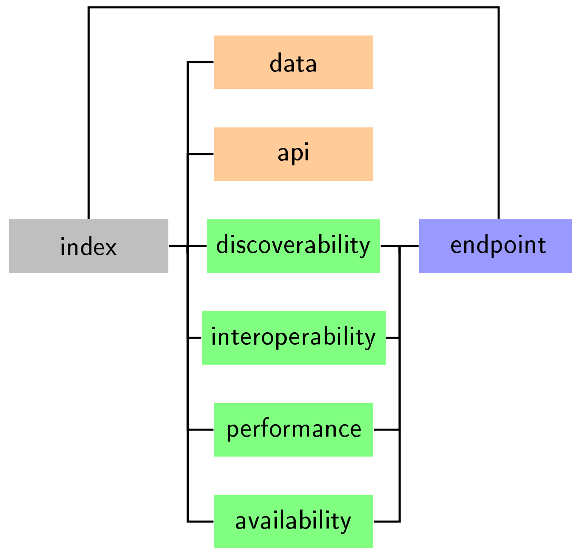


Fig. 2. SPARQL user-interface sitemap

DISCOVERABILITY provides the server name, VoID locations and SD availability found (if any) for each endpoint in the most recent experiment.

INTEROPERABILITY counts the SPARQL 1.0 and SPARQL 1.1 queries passed by each endpoint in the most recent experiment.

PERFORMANCE provides the mean performance for join and ask queries in cold/warm runs from the most recent experiment for all endpoints.

The APIs are provided and described online at: `http://sparqles.ai.wu.ac.at/api`.

5.3. User Interface

The SPARQLES user interface – publicly available at `http://sparqles.ai.wu.ac.at` – offers an entry point for human users interested in the experimental results gathered. The interface is implemented using various Javascript libraries, including Node.js and nvd3 for rendering interactive visualisations.

The homepage offers "at-a-glance" aggregated views of the four dimensions computed across all endpoints. In Figure 3, for example, we see the aggregate view for availability, which shows the evolution of the number of endpoints falling into five different availability intervals ([0–5[, [5–75[, [75–95[, [95–99[, [99-100]). Other aggregate views likewise provide an overview of performance, interoperability and discoverability.

From the homepage, the user has a number of possible navigation steps, as illustrated in Figure 2.

The user can navigate to a page dedicated to each dimension to get an overview of key results for all endpoints in a list view, as follows:

AVAILABILITY lists the availability for each endpoint over the past 24 hours and the past 7 days.

PERFORMANCE lists the suspected result-size threshold and the median cold/warm run-times for `ASK` and `JOIN` queries over the last 10 runs.

INTEROPERABILITY lists the ratio of SPARQL 1.0 and SPARQL 1.1 query test-cases passed by each endpoint in the most recent run.

DISCOVERABILITY indicates whether or not a VoID and/or SD description is available for each endpoint in some location, and what server name could be found (if any), in the most recent run.

Otherwise, either by using the auto-complete search function (available on all pages), or by clicking on a specific endpoint mentioned in one of the previous four list-view pages, the user can arrive to an endpoint-specific view with detailed information about all four dimensions for a given endpoint. An example for the main DBpedia endpoint is provided in Figure 4 & 5 (referring to the same page but split here for formatting purposes). The views provide information on the weekly availability for the past year, median performance for the past ten runs, the interoperability test queries failed or passed[18] as well as the locations in which VoID/SD descriptions can be found and the server name detected (if any).

From the homepage, there are also links to download data dumps or view the A.P.I. documentation.

6. Evaluation

SPARQLES has been running since October 2013 where, at the time of writing, it currently monitors 535 endpoints. We now present some high-level results with respect to operating, maintaining and improving various aspects of the SPARQLES system.

Analytics With respect to running the individual analytical tasks, we measured the time of the four most recent runs within the SPARQLES system.

The fastest were the AVAILABILITY experiments, which took between 41–44 minutes ($\mu = 41.8$ minutes,

[18] On hovering the mouse pointer over the query name, the user can see the full query, and on hovering the pointer over a red icon, the user can see details the exception encountered.

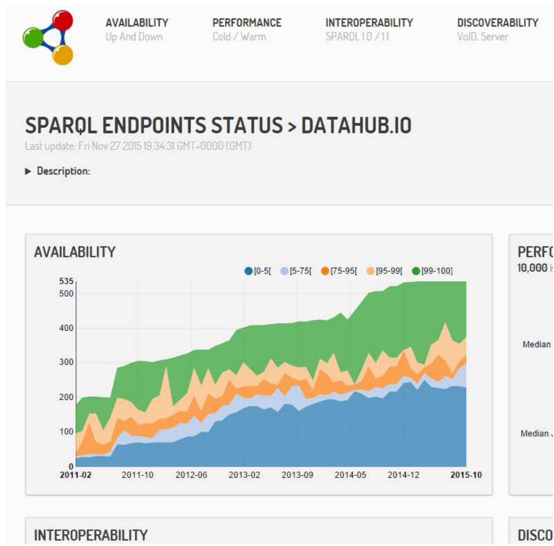


Fig. 3. Screen capture of SPARQLES online system's homepage focusing on availability overview. The homepage offers "at-a-glance" aggregated views for the four dimensions of analytics described.

σ = 1.1 minutes) in the four most recent runs. These experiments involve sending either one or two simple queries to each server. This runtime fits within the current one hour interval; however, if we were to increase the number of endpoints observed by about 33%, we would need to increase the interval to (say) two hours, to ensure that the analytics terminate in time.

Second were the DISCOVERABILITY experiments, which took between 65–67 minutes (μ = 65.5 minutes, σ = 1 minute) in the most recent four runs. These experiments involve sending a single query and checking two Web documents for metadata about each endpoint. These experiments could easily be maintained at the weekly interval, even if the number of endpoints observed were to increase, e.g., a hundredfold.

Third were the INTEROPERABILITY experiments, which took between 65–87 minutes (μ = 72.6 minutes, σ = 9 minutes) in the four most recent runs. These experiments involve running 42 queries against each remote endpoints. However, since we have refactored these queries to return empty results, they should be trivial to run. These experiments are likewise easily maintained within their current weekly interval.

Fourth were the PERFORMANCE experiments, which took between 537–614 minutes (μ = 583.5 minutes, σ = 28.8 minutes) in the four most recent runs. Against each endpoint, we run 17 queries twice (once cold and once warm). These experiments take significantly longer than the others: although INTEROPERABILITY has more queries, the queries in PERFORMANCE are designed to be non-trivial to answer, returning thousands of results and requiring a significant amount of processing for the endpoint to service. Within the current daily interval and settings, we could support these analytics for about twice the number of endpoints.

Storage An important question for the sustainability of SPARQLES is how its storage requirements grows over time: we not only add information about each endpoint, but also add new endpoints. Along these lines, Figure 6 presents the sizes of weekly compressed backup dumps of the MongoDB store captured over a 22 week period in 2015 (from April 20th to September 14th). We see that the dump sizes grew on average by 3.5 megabytes per month. The most recently compressed dump – 397 megabytes – corresponds to a live MongoDB index of 14 gigabytes (approximately 35.25$\times$ larger than the dump). Thus although the index size growth is accelerating, we still have considerable time before, for example, the SPARQLES service would fill a standard 1 terabyte harddrive; at that stage, we would need to consider aggregating and/or archiving older experimental data.

A.P.I. In order to ensure that our A.P.I.'s would perform well under high loads, we sent 1,000 requests to each of our A.P.I.'s from 10, 50 and 100 parallel clients respectively. Figure 7 provides box-plots of the individual runtimes encountered. Under this type of load, although we see that the slowest request can range in the tens of seconds (especially for a higher number of clients), typical performance in the lower three quartiles remains reliably below half a second. In summary, we encounter a few slow requests that require up to 20 seconds to complete, but the majority of requests are answered within 0.5 seconds, even with 100 clients simultaneously issuing 1,000 requests.

Usability We are constantly collecting and reacting to user feedback relating to bugs, feature-requests and usability on the issue tracker for the project available on `https://github.com/pyvandenbussche/sparqles/issues` (which at the time of writing contains 21 open issues and 24 closed issues). Feedback has related to varied aspects, be it misreported statistics for the endpoints maintained by users, or the expense of certain queries for remote services, or problems with characters/escaping in the interface, or requests for various enhancements. This feedback has been invaluable for improving the usability, correctness and sustainability of the service. For example, one

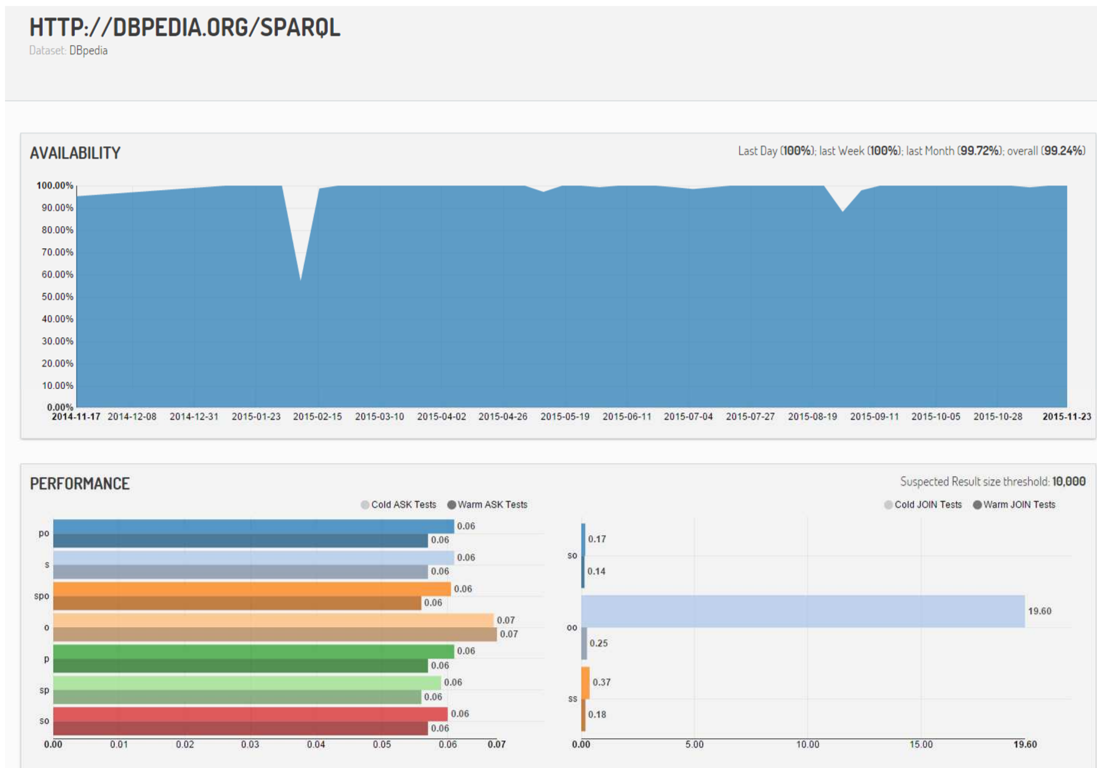


Fig. 4. Screen capture of the detailed view about the DBpedia endpoint (`http://dbpedia.org/sparql`) showing the results of availability for the past year and performance tests based on the median result for the most recent ten runs.

of the most important changes we have made based on this feedback was to modify the INTEROPERABILITY queries to reduce the computational strain they were placing on the public endpoints that SPARQLES monitors. We are continuing to address the open issues and consider the other feedback and requests for enhancements that we receive from the community.

7. Discussion

We now discuss the use-cases, impact thus far, limitations and sustainability of SPARQLES.

7.1. Impact

One of the main goals of the system is to disseminate timely information about the health of individual endpoints. To help characterise the impact of SPARQLES, in the following we present some statistics collected from the Google Analytics for the site.

Over a 23 month period, SPARQLES has seen a total of 11,420 user sessions, averaging about 497 sessions per month. Figure 8 presents the data for the past 23 months, where we see a peak in October 2013, after which the number of user sessions was between 172 (February 2014) and 823 (June 2015). In Table 1, we present the number of sessions broken down by the most visited first page, second page, and third page.[19] The table provides details on the first page the visitors access, followed by the page accessed from the 1st and 2nd interaction. In general, we see that of the four dimensions analysed, users are most concerned with availability. We also found that the most common user interaction involves starting on the homepage, traversing to the availability overview, and then onto the detailed view of a specific endpoint: this can be seen from the first (non-total) data row of Table 1.

Another indirect goal of the system is to encourage endpoints to follow best practices: we would hope

[19]Please note that since we cannot access raw data, some of the figures may be rounded (for example, the Google Analytics system reports "`3.4K`" rather than an exact figure). Likewise we only have details of visits to the top twenty pages, hence we may only have an upper-bound for other pages.

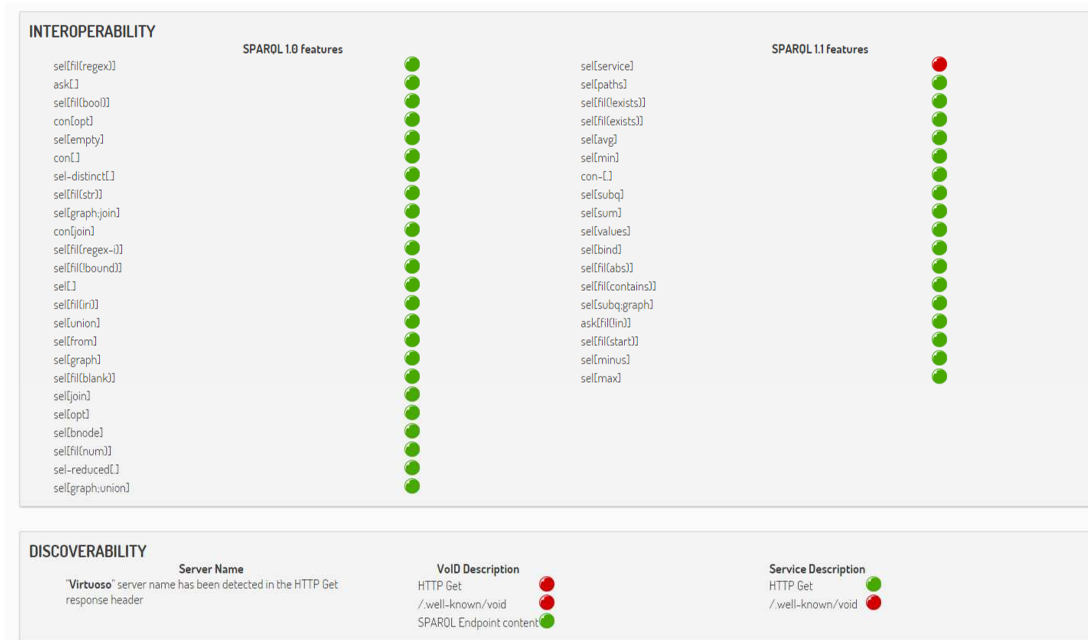


Fig. 5. Screen capture of the detailed view about the DBpedia endpoint (`http://dbpedia.org/sparql`) showing the results of interoperability (individual queries passed or failed) and discoverability tests (locations of VoID and SD descriptions) in the most recent runs.

Table 1

Number of SPARQLES sessions over a 22 month period spanning from 2013/09/27–2015/07/27.

START PAGE		1ST INTERACTION		2ND INTERACTION	
Total	**11,420**	**Total**	**3,400**	**Total**	**2,300**
`/`	8,300	`/availability`	1,500	*specific endpoint*	1,202
specific endpoint	1,247	*specific endpoint*	949	`/availability`	340
`/availability`	1,200	`/discoverability`	286	`/`	326
`/discoverability`	221	`/interoperability`	214	`/interoperability`	179
`/interoperability`	172	`/performance`	201	`/performance`	151
`/performance`	45	`/`	133	`/discoverability`	126
`/api`	16	`/api`	27	`/api`	<12
`/data`	<11	`/data`	<7	`/data`	<12

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,[20] it is difficult to ascertain to what degree SPARQLES has had an impact on the maturity of SPARQL endpoints in this respect.

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

[20] See for example `https://github.com/pyvandenbussche/sparqles/issues/42`.

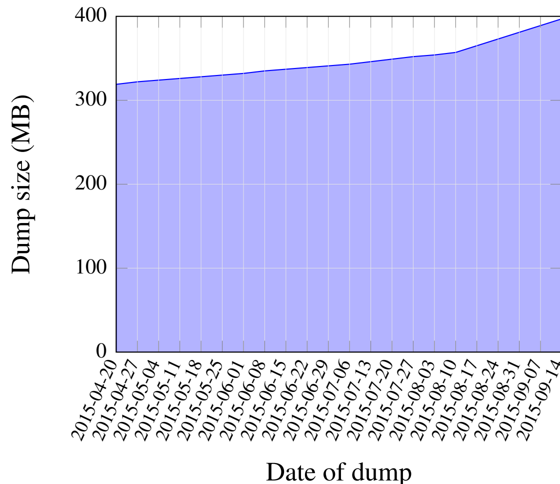


Fig. 6. Evolution of data-volumes spanning 22 weeks

costs – which draws heavily upon the availability statistics from our original analysis to justify why alternatives to SPARQL are needed [35,34]. 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. [13] 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]. Aside from this, we can point to a number of papers explicitly using the SPARQLES system itself, including works by Benedetti et al. [5], Atemezing & Troncy [4], Rietveld & Hoekstra [30], etc. At the time of writing, according to Google Scholar, the original paper describing our original experiments [9] has received 99 citations over the past two years.[21]

7.2. Limitations

For each of the analytics presented in Section 4, 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.*

[21] `https://scholar.google.com/citations?view_op=view_citation&hl=en&citation_for_view=CP-fgY4AAAAJ:RHpTSmoSYBkC`

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 but it is very much possible.

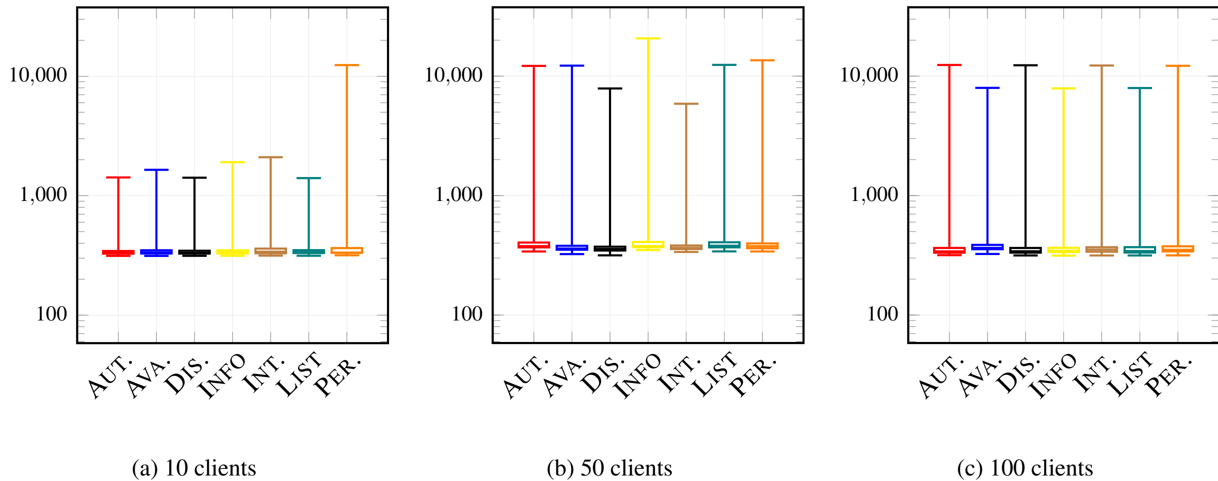
Second, as a more pragmatic issue, since we first put the system online in November 2013, we have had various local reliability issues, where data were not collected for certain weeks, where data were lost due to server migration, and where the site itself was offline. During this period, we have been 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 years.

Third, some of the analytics may be biased towards servers that are closer geographically to the SPARQLES host in Austria. One option to mitigate this bias – as well as local reliability issues – would be to replicate SPARQLES analytics in multiple remote locations and create a mechanism for aggregating a global consensus across all remote instances. Currently we do not have the resources available to host another instance of SPARQLES. However, the SPARQLES code is available for download, where the community can download and install their own instances, perhaps targeted at those endpoints of interest to them.

7.3. Sustainability

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 revised the interoperability queries to make them less costly and have been attentive in addressing all complaints raised in our issue tracker relating to the cost SPARQL puts on remote servers.

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 cre-



(a) 10 clients (b) 50 clients (c) 100 clients

Fig. 7. A.P.I. response times for 1,000 requests with 10, 50 and 100 parallel clients. On the x-axis, AUT. denotes Autocomplete, AVA. denotes Availability, DIS. denotes Discoverability, INT.. denotes Interoperability, PERF.. denotes performance. The y-axis shows the response times in milliseconds; the axis is given in log scale and aligned horizontally for comparison across the three plots. The box plots are drawn for the maximum, third quartile (75th percentile), median, first quartile (25th percentile) and minimum times from the individual responses.

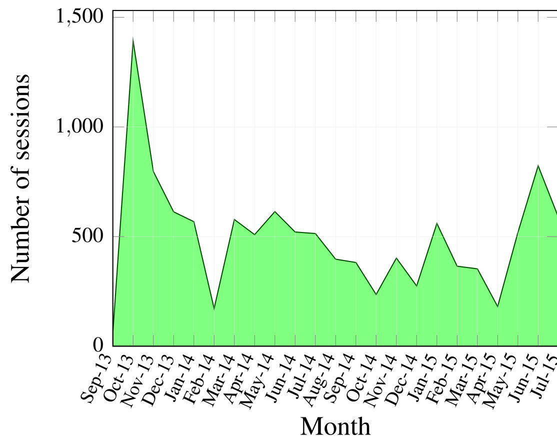


Fig. 8. Number of unique sessions per month for SPARQLES

ating Linked Data IRIs 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. There are numerous directions in which SPARQLES could still be improved, which we will tackle as time progresses.

8. 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 A.P.I.'s for the collected results. We presented four dimensions of analytics that the system runs over public endpoints. Thereafter, we presented some of the details of the interfaces SPARQLES provides for human and automated agents to interact with the underlying data collection. We also presented some measures with respect to the overall runtime of analytics, the growth in storage requirements, and the performance of our A.P.I.'s under load. 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 and quite critical perspective on public SPARQL endpoints. 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 thank the Open Knowledge Foundation for agreeing to host the project for over a year.

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 Posters & Demos*, 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] G. A. Atemezing and R. Troncy. Towards a linked-data based visualization wizard. In *PInternational Workshop on Consuming Linked Data (COLD 2014)*, 2014.

[5] F. Benedetti, S. Bergamaschi, and L. Po. Online index extraction from linked open data sources. In *Second International Workshop on Linked Data for Information Extraction (LD4IE 2014)*, pages 9–20, 2014.

[6] B. Berendt, L. Hollink, V. Hollink, M. Luczak-Rösch, K. Möller, and D. Vallet. Usage analysis and the web of data. *SIGIR Forum*, 45(1):63–69, 2011.

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

[8] 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.

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

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

[11] I. Ermilov, M. Martin, J. Lehmann, and S. Auer. Linked open data statistics: Collection and exploitation. In *Knowledge Engineering and the Semantic Web (KESW)*, pages 242–249, 2013.

[12] L. Feigenbaum, G. T. Williams, K. G. Clark, and E. Torres. SPARQL 1.1 protocol. W3C Recommendation, March 2013.

[13] 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.

[14] 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.

[15] P. Gearon, A. Passant, and A. Polleres. SPARQL 1.1 update. W3C Recommendation, March 2013.

[16] B. Glimm and C. Ogbuji. SPARQL 1.1 update. W3C Recommendation, March 2013.

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

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

[19] 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/.

[20] 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(2):167–195, 2015.

[21] J. Lorey. Identifying and determining SPARQL endpoint characteristics. *IJWIS*, 10(3):226–244, 2014.

[22] M. Mehdi, A. Iqbal, A. Hogan, A. Hasnain, Y. Khan, S. Decker, and R. Sahay. Discovering Domain-Specific Public SPARQL Endpoints: A Life-Sciences Use-Case. In *IDEAS*, 2014.

[23] H. Paulheim and S. Hertling. Discoverability of SPARQL Endpoints in Linked Open Data. In *ISWC (Posters & Demos)*, pages 245–248, 2013.

[24] 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.

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

[26] E. Prud'hommeaux and C. Buil-Aranda. SPARQL 1.1 federated query. W3C Recommendation, March 2013.

[27] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, January 2008.

[28] S. Ran. A model for web services discovery with qos. *SIGecom Exchanges*, 4(1):1–10, 2003.

[29] L. Rietveld and R. Hoekstra. Man vs. machine: Differences in SPARQL queries. In *USEWOD workshop*, 2014.

[30] L. Rietveld and R. Hoekstra. The YASGUI Family of SPARQL Clients. *Semantic Web Journal*, 2015. (under review).

[31] M. Saleem, M. I. Ali, A. Hogan, Q. Mehmood, and A. N. Ngomo. LSQ: the linked SPARQL queries dataset. In *ISWC*, pages 261–269, 2015.

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

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

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

[35] 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.

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