# Linked Data-as-a-Service: The Semantic Web Redeployed

Laurens Rietveld[1(✉)], Ruben Verborgh[2], Wouter Beek[1(✉)], Miel Vander Sande[2], and Stefan Schlobach[1(✉)]

[1] Department of Computer Science, VU University Amsterdam, Amsterdam, The Netherlands
{laurens.rietveld,w.g.j.beek,stefan.schlobach}@vu.nl
[2] Ghent University – IMinds, Ghent, Belgium
{ruben.verborgh,miel.vandersande}@ugent.be

**Abstract.** Ad-hoc querying is crucial to access information from Linked Data, yet publishing queryable RDF datasets on the Web is not a trivial exercise. The most compelling argument to support this claim is that the Web contains hundreds of thousands of data documents, while only 260 queryable SPARQL endpoints are provided. Even worse, the SPARQL endpoints we *do* have are often unstable, may not comply with the standards, and may differ in supported features. In other words, hosting data online is easy, but publishing Linked Data via a queryable API such as SPARQL appears to be too difficult. As a consequence, in practice, there is no single uniform way to query the LOD Cloud today. In this paper, we therefore combine a large-scale Linked Data publication project (LOD Laundromat) with a low-cost server-side interface (Triple Pattern Fragments), in order to bridge the gap between the Web of downloadable data documents and the Web of live queryable data. The result is a repeatable, low-cost, open-source data publication process. To demonstrate its applicability, we made over 650,000 data documents available as data APIs, consisting of 30 billion triples.

**Keywords:** API · Data publishing · Web Services · Linked Data

## 1 Introduction

In 2001 the Semantic Web promised to provide a distributed and heterogeneous data space, like the traditional Web, that could at the same time be used as a machine-readable Web Services platform [4]. Data publishers would open up their knowledge for potentially unanticipated reuse by data consumers. Intelligent agents would navigate this worldwide and heterogeneous data space in order to perform intelligent tasks. In 2015 this promise remains largely unmet.

When we look at empirical data about the rudimentary infrastructure of the Semantic Web today, we see multiple problems: Millions of data documents exist

This work was supported by the Dutch national program COMMIT.

© Springer International Publishing Switzerland 2015
F. Gandon et al. (Eds.): ESWC 2015, LNCS 9088, pp. 471–487, 2015.
DOI: 10.1007/978-3-319-18818-8_29

that potentially contain information that is relevant for intelligent agents. However, only a tiny percentage of these data documents can be straightforwardly used by software clients. Typically, online data sources cannot be consistently queried over a prolonged period of time, so that no commercial Web Service would dare to depend on general query endpoint availability and consistency. In practice, Semantic Web applications run locally on self-deployed and centralized triple-stores housing data that has been integrated and cleaned for a specific application or purpose. Meanwhile, the universally accessible and automatically navigable online Linked Open Data (LOD) Cloud remains structurally disjointed, unreliable, and — as a result — largely unused for building the next generation of large-scale Web solutions.

The problem here is *sustainability*. While it is technically possible to publish data in a standards-compliant way, many data publishers are unable to do so. While it is technically possible to pose structured live queries against a large dataset, this is prohibitively expensive in terms of both engineering effort and hardware support.

Take for instance the concept of federation, in which a query is evaluated against multiple datasets at the same time. According to the original promise of the Semantic Web federation is crucial, since it allows an automated agent to make intelligent decisions based on an array of knowledge sources that are both distributed and heterogeneous. In practice, however, federation is extremely difficult [19] since most datasets do not have a live query endpoint; the few query endpoints that do exist often have low availability; the few available live query endpoints sometimes implement constrained APIs which makes it difficult to guarantee that queries are answered in a consistent way.

We have performed a redeployment of the LOD Cloud that makes the Semantic Web queryable on an unprecedented scale, while retaining its originally defined properties of openness and heterogeneity. We provide an architecture plus working implementation which allows queries that span a large number of heterogeneous datasets to be performed. The working implementation consists of a full-scale and continuously updating copy of the LOD Cloud as it exists today. This complementary copy can be queried by intelligent agents, while guaranteeing that an answer will be established consistently and reliably. We call this complementary copy *Linked Data-as-a-Service* (LDaaS).

LDaaS was created by tightly combining two existing state-of-the-art approaches: the LOD Laundromat and Linked Data Fragments. While the integration itself is straightforward, we show that its consistent execution delivers a system that is able to meet a wide-spanning array of requirements that have not been met before in both width and depth.

This paper is organized as follows: Sect. 2 given an overview of the core concepts and related work. Section 3 details the motivation behind LDaaS. Section 4 specifies the architecture and design of LDaaS, which we evaluate in Sect. 5. We conclude in Sect. 6.

## 2 Core Concepts and Related Work

### 2.1 Web Interfaces to RDF Data

In order to characterize the many possibilities for hosting Linked Datasets on the Web, *Linked Data Fragments* (LDF)[26] was introduced as a uniform view on all possible Web APIs to Linked Data. The common characteristic of all interfaces is that, in one way or another, they offer specific parts of a dataset. Consequently, by analyzing the parts offered by an interface, we can analyze the interface itself. Each such part is called a *Linked Data Fragment*, consisting of:

- **data:** the triples of the dataset that match an interface-specific *selector*;
- **metadata:** triples that describe the fragment;
- **controls:** hyperlinks and/or hypermedia forms that lead to other fragments.

The choices made for each of those elements influence the functional and non-functional properties of an interface. This includes the effort of a server to generate fragments, the cacheability of those fragments, the availability and performance of query execution, and the party responsible for executing those queries.

Using this conceptual framework, we will now discuss several interfaces.

**Data Dumps.** File-based datasets are conceptually the most simple APIs: the *data* consists of all triples of the dataset. They are possibly combined into a compressed archive and published at a single URL. Sometimes the archive contains *metadata*, but *controls*—with the possible exception of HTTP URIs in RDF triples—are not present. Query execution on these file-based datasets is entirely the responsibility of the client; obtaining up-to-date query results requires re-downloading the entire dataset periodically or upon change.

**Linked Data Documents.** By organizing triples by subject, Linked Data Documents allow to *dereference* the URL of entities. A document's *data* consists of triples related to the entity (usually triples where the subject or object is that entity). It might contain *metadata* triples about the document (e.g. creator, date) and its *controls* are the URLs of other entities, which can be dereferenced in turn. Linked Data Documents provide a fast way to collect the authoritative information about a particular entity and they are cache-friendly, but predicate- or object-based queries are practically infeasible.

**SPARQL Endpoints.** The SPARQL query language [13] allows to express very precise selections of triples in RDF datasets. A SPARQL endpoint [10] allows the execution of SPARQL queries on a dataset through HTTP. A fragment's *data* consists of triples matching the query (assuming the `CONSTRUCT` form); the *metadata* and *control* sets are empty. Query execution is performed entirely by the server, and because each client can ask highly individualized requests, the cacheability of SPARQL fragments is quite low. This, combined with complexity

of SPARQL query execution, likely contributes to the low availability of public SPARQL endpoints [1,7]. To mitigate this, many endpoints restrict usage, by reducing the allowed query execution time, limiting the number of rows that can be returned or sorted, or not supporting more expensive SPARQL features [7].

**Triple Pattern Fragments.** The Triple Pattern Fragments (TPF) API [25] has been designed to minimize server processing, while at the same time enabling efficient live querying on the client side. A fragment's *data* consists of all triples that match a specific triple pattern, and can possibly be paged. Each fragment (page) contains the estimated total number of matches, to allow for query planning, and contains hypermedia controls to find all other Triple Pattern Fragments of the same dataset. The controls ensure each fragment is *self-describing*: just like regular webpages do for humans, fragments describes in a machine-interpretable way what the possible actions are and how clients can perform them. Consequently, clients can use the interface without needing the specification. Complex SPARQL queries are decomposed by clients into Triple Pattern Fragments. Since requests are less granular, fragments are more likely to be reused across clients, improving the benefits of caching [25]. Because of the decreased complexity, the server does not necessarily require a triple-store to generate its fragments.

**Other Specific APIs.** Several APIs with custom fragments types have been proposed, including the Linked Data Platform [24], the SPARQL Graph Store Protocol [20], and other HTTP interfaces such as the Linked Data API [17] and Restpark [18]. In contrast to Triple Pattern Fragments, the fragments offered by these APIs are not self-describing: clients require an implementation of the corresponding specification in order to use the API, unlike the typically self-explanatory resources on the human Web. Furthermore, no query engines for these interfaces have been implemented to date.

### 2.2 Existing Approaches to Linked Data-as-a-Service

**Large Linked Datasets.** The Billion Triple Challenge[1] is a collection of crawled Linked Data that is publicly available and that is often used in Big Data research. It is crawled from the LOD Cloud [5] and consists of 1.4 billion triples. It includes large RDF datasets, as well as data in RDFa and Microformats. However, this dataset is not a complete crawl of the LOD Cloud (nor does it aim to be), as datasets from several catalogs are missing. Additionally, the latest version of this dataset dates back to 2012.

Freebase [6] publishes 1.9 billion triples, taken from manual user input and existing RDF and Microformat datasets. Access to Freebase is possible through an API, through a (non-SPARQL) structured query language, and as a complete dump of N-Triples. However, these dumps include many non-conforming, syntactically incorrect triples.

[1] See http://km.aifb.kit.edu/projects/btc-2012/.

**Large-Scale Linked Data Indexes.** In order to make Linked Data available through a centralized interface, Sindice [21], active from 2007 to 2014, crawled Linked Data resources, including RDF, RDFa and Microformats. Datasets were imported on a per-instance and manual opt-in basis. Raw data versions cannot be downloaded and access is granted through a customized API.

LODCache[2], provided by OpenLink, similarly crawls the Web for Linked Data, but does not make data dumps available. Its SPARQL endpoint suffers from issues such as low availability, presumably related to its enormous size of more than 50 billion triples. There is no transparent procedure to include data manually or automatically. Given the focus on size, its main purpose is likely to showcase the scalability of the Virtuoso triple-store, rather than providing a sustainable model for Linked Data consumption on the Web. Other initiatives, such as Europeana [14], aggregate data from specific content domains, and allow queries through customized APIs.

Finally, DyLDO [16] is a long-term experiment to monitor the dynamics of a core set of 80 thousand Linked Data documents on a weekly basis. Each week's crawl is published as an N-Quads file. This work provides interesting insight in how Linked Data evolves over time. It is not possible to easily select triples from a *single* dataset, and not all datasets belonging to the Linked Data Cloud are included. Another form of incompleteness stems from the fact that the crawl is based on URI dereferencing, not guaranteeing datasets are included in their entirety.

**LOD Laundromat.** The LOD Laundromat [3] crawls the Linked Data Cloud, and re-publishes any Linked Dataset it finds, in a canonical, standards-compliant, compressed, N-Triples or N-Quads format. The goal of LOD Laundromat is not that of a primary publication platform. Instead, it is a complementary approach to existing efforts, to publish siblings of existing idiosyncratic datasets. The collection of datasets that it comprises is continuously being extended, both in an automated fashion as well as a manual fashion: anyone can add their dataset URL to the LOD Laundromat[3], where their dataset will be cleaned and re-published. Human data consumers are able to navigate a large collection of high-quality datasets, and download the corresponding clean data. Additionally, machine processors are able to easily load very large amounts of real-world data, by selecting clean data documents through a SPARQL query.

**Dydra.** Dydra[4] is a cloud-based RDF graph database, which allows users without hosting capabilities to publish RDF graphs on the Web. Via their Web interface, Dydra provides a SPARQL endpoint, the option to configure permissions, and other graph management features. However, access to Dydra is limited: free access is severely restricted, and there are no public pay plans for paid services.

[2] See http://lod.openlinksw.com/.
[3] See http://lodlaundromat.org/basket.
[4] See http://dydra.com.

## 3 Motivation

In this section, we motivate why there is a need for an alternative deployment of the Semantic Web, and why we opt for a Linked Data-as-a-Service approach.

### 3.1 Canonical Form

One of the biggest hurdles towards Web-scale live querying is that — at the moment — Semantic Web datasets cannot all be queried in the same, uniform way (Problem 1).

**Problem 1.** *In practice, there is no single, uniform way in which the LOD Cloud can be queried today.*

First of all, most Semantic Web datasets that are available online are data dumps [9,15], which implies that they cannot be queried live. In order to perform structured queries on such datasets, one has to download the data dumps and deploy them locally. Secondly, many data dumps that are available online are not fully standards-compliant [2,15]. This makes the aforementioned local deployment relatively difficult, since it requires the use of tools that can cope with archive errors, HTTP errors, multiple syntax formats, syntax errors, etc. Thirdly, not all datasets that *can* be queried live use a standardized query language (such as SPARQL). Indeed, some require a data consumer to formulate a query in a dedicated query language or to use a custom API. Fourthly, most custom APIs are not self-describing, making it relatively difficult for a machine processor to create such queries on the fly. Fifthly, most online datasets that can be queried live and that are using standardized query languages such as SPARQL are imposing restrictions on queries that can be expressed and results that can be returned [1,7]. Finally, different SPARQL endpoints impose *different* restrictions [7]. This makes it difficult for a data consumer to predict whether, and if so how, a query will be answered. The latter point is especially relevant in the case of federated querying (see Sect. 3.4), where sub-queries are evaluated against multiple endpoints with potentially heterogeneous implementations.

For the last decade or so, Problem 1 has been approached by creating standards, formulating guidelines, and building tools. In addition, Semantic Web evangelists have tried to educate and convince data producers to follow those guidelines and use those tools. This may still be the long-term solution. However, we observe that this approach has been taken for over a decade, yet leading to the heterogeneous deployment described above. We therefore introduce the complementary Solution 1 that allows all Semantic Web data to be queried live in a uniform way and machine-accessible way.

**Solution 1.** *Allow all Semantic Web documents to be queried through a uniform interface that is standards-compatible and self-descriptive.*

### 3.2 Scalability and Availability

After the first 14 years of Semantic Web deployment there are at least millions of data documents [8,12] but only 260 live query endpoints [7]. Even though the number of endpoints is growing over time [1,7], at the current growth rate, the gap between data dumps and live queryable data will only increase (Problem 2). The number of query endpoints remains relatively low compared to the number of datasets, and many of the endpoints that do exist suffer from limited availability [7].

**Problem 2.** *Existing deployment techniques do not suffice to close the gap between the Web of downloadable data documents and the Web of live queryable data.*

Several causes contribute to Problem 2. Firstly, it is difficult to deploy Semantic Web data, since this currently requires a complicated stack of software products. Secondly, existing query endpoints perform most calculations on the server-side, resulting in a relatively high cost and thus a negative incentive for the data publisher. Thirdly, in the presence of dedicated query languages, custom APIs, and restricted SPARQL endpoints, some have advocated to avoid SPARQL endpoints altogether, recommending the more flexible data dumps instead, thereby giving up on live querying. Solution 2 addresses these causes.

**Solution 2.** *Strike a balance between server- and client-side processing, and automatically deploy all Semantic Web data as live query endpoints. If clients desire more flexibility, they can download the full data dumps as well.*

### 3.3 Linked Data-as-a-Service

Even though software solutions exist to facilitate an easy deployment of various Web-related services such as email, chat, file sharing, etc., in practice users gravitate towards centralized online deployments (e.g., Google and Microsoft mail, Facebook chat, Dropbox file sharing). We observe similar effects in the (lack of) popularization of Semantic Web technologies (Problem 3). Even though multiple software solutions exist for creating, storing, and deploying Semantic Web services (e.g., RDF parsers, triple-stores, SPARQL endpoints), empirical observations indicate that the deployment of such services with existing solutions has been problematic [15]. As a consequence, live querying of Semantic Web data has not yet taken off in the same way as other Web-related tasks have.

**Problem 3.** *Even though a technology stack for publishing Semantic Web data exists today, there is currently no simplified Web Service that does the same thing on a Web-scale.*

While technologies exist that make it possible to publish a live query endpoint over Semantic Web data, there is currently no simplified Web *Service* that allows data to be deployed on a very large scale. Under the assumption that take-up of traditional Web Services is an indicator of future take-up of Semantic Web

Services (an assumption that cannot be proven, only argued for), it follows that many data publishers may prefer a simplified Web Service to at least perform some of the data publishing tasks (Solution 3).

**Solution 3.** *Provide a service to take care of the tasks that have proven to be problematic for data publishers, having an effective cost model for servicing a high number of data consumers.*

### 3.4 Federation

In a federated query, sub-queries are evaluated by different query endpoints. For example, one may be interested in who happens to know a given person by querying a collection of HTML files that contain FOAF profiles in RDFa. At present, querying multiple endpoints is problematic (Problem 4), because of the cumulating unavailability of individual endpoints, as well as the heterogeneity of interfaces to Linked Data.

**Problem 4.** *On the current deployment of the Semantic Web it is difficult to query across multiple datasets.*

Given the heterogeneous nature of today's Semantic Web deployment (Sect. 3.1), there are no LOD Cloud-wide guarantees as to whether, and if so how, sub-queries will be evaluated by different endpoints. In addition, properties of datasets (i.e., metadata descriptions) may be relevant for deciding algorithmically which datasets to query in a federated context. Several initiatives exist that seek to describe datasets in terms of Linked Data (e.g., VoID, VoID-ext, Bio2RDF metrics, etc.). However, such metadata descriptions are often not available, and oftentimes do not contain enough metadata in order to make efficient query federation possible.

**Solution 4.** *Allow federated queries to be evaluated across multiple datasets. Allow metadata descriptions to be used in order to determine which datasets to query.*

## 4 Workflow and Architectural Design

The scale of the LOD Cloud requires a low-cost data publishing workflow. Therefore, the LOD Laundromat service is designed as a (re)publishing platform for data dumps, i.e. data files. As detailed in Sect. 2.1, data dumps are the most simple API that can be offered. To allow structured live querying, while still maintaining technical and economical scalability, we have integrated the low-cost TPF API.

We first discuss the publishing workflow supported by the combination of the LOD Laundromat and Triple Pattern Fragments. We then elaborate on the architectural design of their integration, and how we improved both approaches to keep LDaaS scalable.

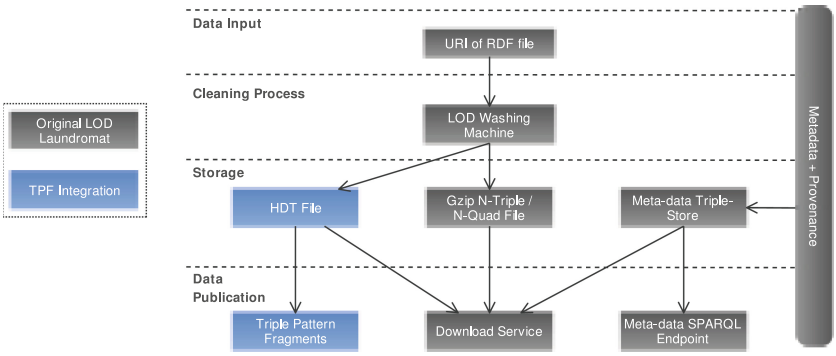


**Fig. 1.** LOD Laundromat (re)-publishing workflow

### 4.1 Re-Publishing Workflow

Figure 1 shows the re-publishing workflow of the LOD Laundromat (already presented in [3]), extended with the Triple Pattern Fragment API. Here, we see how the LOD Laundromat (1) takes a reference to an online RDF document as input, (2) cleans the data in the LOD Washing Machine, (3) stores several representations of the data, and publishes the data through several APIs. Below, we discuss each step of the workflow in detail.

**Data Input.** The LOD Laundromat maintains a collection of Linked Data seed points, called the LOD Basket[5]. The seed points are Web locations from which generally non-standards compliant or 'dirty' data can be downloaded.

**Cleaning Process.** The LOD Washing Machine[6] is the module of the LOD Laundromat that takes 'dirty' data documents from the LOD Basket and tries to download them. Potential HTTP errors are stored as part of the data document's metadata description that is generated by the LOD Washing Machine. Data documents that occur in archives are recursively unpacked. Once fully unpacked, the serialization format of the data document is determined heuristically based on the file extension (if any), the value of the `Content-Type` HTTP header (if present), and a lenient parse of the first chunk of the data file. All standard RDF 1.1 serialization formats are supported: N-Quads, N-Triples, RDFa, RDF/XML, TriG, and Turtle. Once the serialization format has been guessed, the document is parsed. Since many data documents contain syntax errors[7], only compliant triples are retained. Every warning is stored as metadata of the resultant dataset in order to make the cleaning process transparent. As a final step, VoID descriptions that occur within a document that is being cleaned are added to the LOD Basket for future cleaning.

[5] See http://lodlaundromat.org/basket.
[6] See https://github.com/LODLaundry/llWashingMachine.
[7] For an indication, see http://lodlaundromat.org/visualizations/.

**Storage.** The output of the cleaning process is stored in a canonical and easy to process data format and is compressed using Gzip. The serialization format is either N-Triples or N-Quads (depending on whether or not at least one quadruple is present in the data file). The statements in the file are sorted lexicographically and any duplicates are removed. IRIs are all encoded in the same way and the lexical expressions of data-typed literals are mapped to their canonical lexical form. This means that two lines are guaranteed to denote the same statement iff they compare identically on a character-by-character basis. Because of these properties it is easy to process cleaned data files in a uniform way, e.g., by streaming through a dataset knowing that the next triple ends with the next newline character.

Next to compressed Gzip files, the datasets are stored as 'Header, Dictionary Triples' (HDT) files as well. HDT files are compressed, indexed files, in a binary serialization format. HDT files are suitable for browsing and querying RDF data without requiring to decompression and/or ingestion into a triple-store [11].

Besides the cleaned data files, LOD Laundromat uses a triple-store which disseminates the metadata obtained during the cleaning process. The triple-store also contains metrics about the structural properties of each cleaned data document[8].

**Data Publication.** The LOD Wardrobe[9] module provides several APIs to access the data generated by the LOD Laundromat.

The first API supports complete control over the data: the HDT and cleaned compressed N-Triples/N-Quads files are available for download. An RDF dump of the LOD Laundromat metadata is available for download as well. The HDT files allows users to download datasets and either query them directly on the command-line (via triple-pattern queries), or to publish these via a Triple Pattern Fragment API. The low-level access to the compressed N-Triples/N-Quads files allow bulk processing of such files, particularly considering the advantages that come with this canonical format: streamed processing of a sorted set of statements.

The TPF API provides access via triple pattern queries, and uses HDT files as storage type. This low-cost API, discussed in Sect. 2, enables structured querying on the crawled datasets.

Finally, the third API is a SPARQL endpoint, which provides SPARQL access to the metadata triple-store. The combination of this API with the previous two is powerful: the SPARQL endpoint enables finding datasets based on structural properties such as the in-degree, out-degree, serialization format, etc. Based on the query results, the user can access the datasets by either downloading the Gzip or HDT files, or accessing the Triple Pattern Fragments API.

[8] Under submission: "LOD in a Box: The C-LOD Meta-Dataset". See http://www.semantic-web-journal.net/content/lod-box-c-lod-meta-dataset.

[9] See http://lodlaundromat.org/wardrobe/.

### 4.2 LDaaS Architectural Design

The architecture for crawling cleaning Linked Data is described in [3], where the architecture of TPF is described in [25]. Below we discuss the measures we took to combine both systems, and the improvements we made to the scalability of LDaaS.

**TPF Horizontal Scalability.** The HDT library can read and query HDT files that are larger than main memory by loading them as *memory-mapped files*. This means the file is mapped byte-by-byte to pages of virtual memory of the application, and regions of the file are swapped in and out by the operating system as needed. This is not horizontally scalable though: although this approach works for 30–40 large datasets, in practice, processes with hundreds or thousands of memory-mapped files tend to become unstable.

Therefore, we extended the TPF architecture with an alternative strategy in which only very large HDT files ($\geq 10\,\text{GB}$) are mapped to memory. In order to query the remaining majority of smaller files, an out-of-process approach is used. When an HTTP request arrives, the server spawns an external process that briefly loads the corresponding HDT file, queries the requested triple pattern, and closes it again. While this involves a larger delay than if the file were mapped to the server process, the overhead is limited for such smaller files because of the efficient HDT index format, and it guarantees the server process' stability. The few large files are still memory-mapped, because spawning new processes for them would result in a noticeably longer delay.

**HDT File Generation.** The original LOD Laundromat architecture creates and serves clean compressed Gzipped N-Quads and N-Triples files. To support the use of a TPF API, we extended this implementation by generating HDT files as well. The HDT files are automatically generated based on the clean compressed Gzipped N-Quads and N-Triples files. Because the latest implementation of HDT does not support named graphs, the N-Quads files are processed as regular triples, without the specified graph.

**Adding TPF Datasets Efficiently.** Datasets crawled by the LOD Laundromat should become available via the TPF API in a timely manner. The original TPF API applies several 'sanity checks' on the data documents before hosting them. However, with 650,000 documents in the configuration file, this process requires minutes of processing time. Because the LOD Laundromat pipeline guarantees 'sane' HDT files, we avoid this issue by extending the TPF API with an optimized loading procedure which disables these sanity checks. As a result, re-loading the configuration whenever a new dataset is cleaned, requires seconds instead of minutes.

## 5 Evaluation

We use the architecture described in the previous section to present a working implementation where we publish the LOD Cloud via Triple Pattern Fragments. In this section, we evaluate this deployment and validate the solutions from Sect. 3.

*Solution 1: Allow all Semantic Web documents to be queried through a uniform interface that is standards-compatible and self-descriptive.*

Solution 1 is evaluated analytically. Currently, 650,950 datasets (29,547,904, 444 triples) are hosted as live query endpoints. Although this does not include all existing Semantic Web data, these numbers show that our approach can realistically be applied on Web scale (see Solution 3 for usage numbers).

Since the Triple Pattern Fragments APIs are generated for all data in the LOD Wardrobe, data queryable by LDaaS inherits the completeness and *data* standards-compliance properties of the LOD Laundromat (see [3] for these compliance properties). *Query* standards-compliance — on the other hand — is attained only partially, since the server-centric paradigm of the SPARQL specification is purposefully deviated from in the current approach in order to fulfill Solution 2.[10] This primarily involves those parts of the SPARQL standard that require the Closed World Assumption (something the authors consider to be at odds with the basic tenets of Semantic Web philosophy) and common data manipulation functions that can be easily implemented by a client (e.g., sorting a list, calculating a maximum value).

The Linked Data FragmentsAPI is self-descriptive, employing the Hydra vocabulary for hypermedia-driven Web APIs.[11] Hydra descriptions allow machine processors to detect the capabilities of the query endpoints in an automated way. In addition, the LDaaS query endpoints do not impose restrictions on the number of operations that may be performed or the number of results that can be retrieved. This allows full data graphs to be traversed by machine processors. Also, pagination is implemented in a reliable way, as opposed to SPARQL endpoints which cannot guarantee consistency with shifting `LIMIT` and `OFFSET` statements.

Finally, uniformity is guaranteed on two-levels: data and interface. The former leverages the LOD Laundromat infrastructure (validated in [3]) as an enabler for homogeneous deployment strategies. Thus, when an agent is able to process one data document, it is also able to query 600K+ data documents. The latter denotes that through Triple Pattern Fragments, processing queries only relies on HTTP, the uniform interface of the Web. Queries are processed in exactly the same way by all endpoints, in contrast to the traditional Semantic Web deployment where different endpoints implement different standards, versions or features.

[10] Even though there is a client-side rewriter that allows SPARQL queries to be performed against an LDF server backend, the standards-compliance of this rewriter is not assessed in this paper.

[11] See http://www.hydra-cg.com/spec/latest/core/.

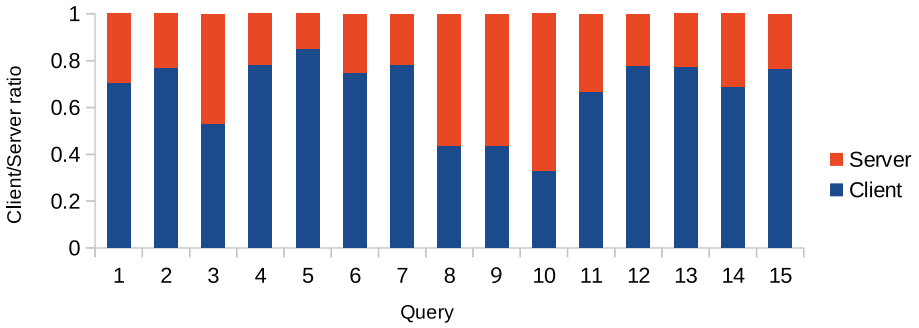


**Fig. 2.** Processing time is shared between client and server.

*Solution 2: Strike a balance between server- and client-side processing, and automatically deploy all Semantic Web data as live query endpoints. If clients desire more flexibility, they can download the full data dumps as well.*

The SPARQL protocol relies on servers to do the heavy lifting: the complete computational processing is performed on the server, and the client is only responsible for sending the request and receiving the SPARQL results. The TPF API, used by LDaaS, takes a different approach. Executing SPARQL queries on the TPF API requires the client to perform joins between triple patterns, and e.g. apply filters or aggregations. As a result, the computational processing is shared between the client and the server, putting less strain on the server.

To quantify this balancing act between server and client-side processing of LDaaS, we evaluated a set of queries from the SP$^2$B benchmark, on a (synthetic) dataset of 10 million triples[12], added to the LOD Laundromat. We measure the client-side and server-side processing time, both running on the same hardware, and excluding network latency. The results, shown in Fig. 2, confirm that the computation is shared between client and server. More specifically, the client does most of the processing for the majority of these SP$^2$B SPARQL queries.

*Solution 3: Provide a service to take care of the tasks that have proven to be problematic for data publishers, having an effective cost model for servicing a high number of data consumers.*

Apart from facilitating common tasks (cleaning, ingesting, publishing), the LOD Laundromat operates under a different cost model than public SPARQL endpoints. In the month prior to submission, the LOD Laundromat served more than 700 users who downloaded 175,000 documents and who issued more than 35,000 TPF API requests.

We consider the hardware costs of disk space and RAM usage below.

*Disk space.* Currently, 650,950 datasets (29,547,904,444 triples) are hosted as Triple Pattern Fragments. The required storage is 265 GB in the compressed

[12] Experiments showed that these results do not differ greatly between SP$^2$B datasets of different sizes.

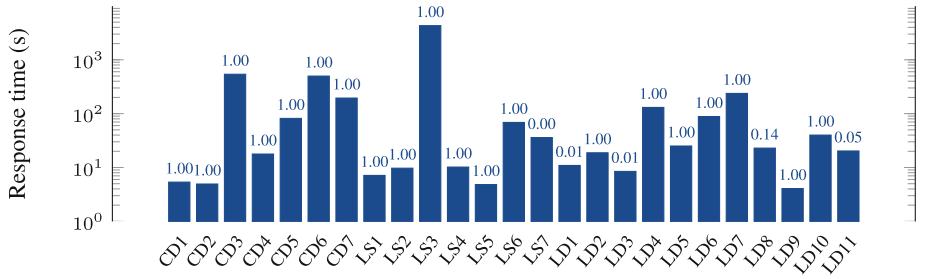


**Fig. 3.** All FedBench queries complete slowly, but successfully, with high average recall (shown on top of each bar) when ran on the deployed LDaaS.

HDT format, or on average 0.41 MB per dataset or 8.97 bytes per triple. The disk space used to store the equivalent gzip-compressed N-Triples (or N-Quads) files is 193 GB (0.30 MB per dataset or 6.53 bytes per triple). Such compressed archives do not allow for efficient triple-pattern queries, which the HDT files can handle at high speed.

*Memory usage.* The TPF server consists of 10 independent worker processes. Because JavaScript is single-threaded, it does not have a concurrency policy for memory access, so each worker needs its own space to allocate resources such as the metadata for each of the 650,950 datasets. However, no further RAM is required for querying or other tasks, since they are performed directly on the HDT files. We have allocated 4 GB per worker process, which was experimentally shown to be sufficient, bringing the total to 40 GB of RAM.

*Solution 4: Allow federated queries to be evaluated across multiple datasets. Allow metadata descriptions to be used in order to determine which datasets to query.*

Finally, we ran FedBench [23] to test the employability of the resulting TPF interfaces for answering federated SPARQL queries. A total of 9 datasets[13], excluding the isolated SP$^2$B dataset, were added to the LOD Laundromat, completing our publishing workflow. Also, we extended the existing TPF client to distribute each fragment request to a predefined list of interfaces and aggregate the results.

We executed the *Cross Domain (CD)*, *Linked Data (LD)*, and *Life Science (LS)* query sets in three runs, directly on http://ldf.lodlaundromat.org from a desktop computer on an external high-speed university network. Figure 3 shows the average execution time for each query with the number of returned results. All queries were successfully completed with an average result recall of 0.81, which confirms the ability to evaluate federated queries. The imperfect recall is a result of an occasional request timeout in queries (LS7, LD1, LD3, LD8, LD11), which, due to limitations of the current implementation, can drop potential

[13] https://code.google.com/p/fbench/wiki/Datasets.

results. Next, general execution time is magnitudes slower compared to state-of-the-art SPARQL Endpoint federation systems [22]. However, this is expected considering *(a)* the LDF paradigm which sacrifices query performance for low server cost, and *(b)* the greedy implementation where the set of sent HTTP requests is a naive Cartesian product between the set of fragments and the datasets.Nevertheless, several queries (LD9, LS5, CD2, CD1, LS1, LD3, LS2) complete within 10 s, which is promising for future development in this area.

## 6 Conclusion

After the first 14 years of Semantic Web deployment the promise of a single distributed and heterogeneous data-space remains largely unfulfilled. Although RDF-based data exists in ever-increasing quantities, large-scale usage by intelligent software clients is not yet a reality. In this paper we have identified and analyzed the main problems that contribute to this lack of usage. Although this list is probably not exhaustive, we selected four pressing problems based on empirical evidence and related work: *(a)* no single uniform way exists to query the LOD cloud; *(b)* there exists a gap between the Web of downloadable data documents and the Web of live queryable data; *(c)* despite the available technology stack, no simplified Web service offers the same functionality on a Web-scale; *(d)* querying across multiple datasets on the current Semantic Web is difficult.

In order to address these issues, we formulated corresponding sustainable solutions, which we proposed and implemented as a redeployment architecture for the Linked Open Data cloud. By combining a large-scale Linked Data publication project (LOD Laundromat) with a low-cost server-side interface (Triple Pattern Fragments), we were able to realize this with minimal engineering.

In doing so, we *(a)* closed the API gap by providing low-cost structured query capabilities to otherwise static datasets; *(b)* did so via a uniform, self-descriptive, and standards-compatible interface; *(c)* enabled in turn federated queries across a multitude of datasets, and *(d)* provide a service for publishers to use. More important than the deployment we provide is the wide applicability of the open source technology stack, whose architecture is detailed in this paper. In contrast to centralized approaches such as the LOD Cache, which focuses on a single centralized database of everything, our approach of one low-cost interface per dataset works in a Web context with multiple servers. It enables querying over multiple datasets by providing clients with the resources needed to perform federation themselves, rather than seizing server-side control of this costly task. To increase accessibility even more, our future work involves disseminating the graph information of N-Quads files via the API as well, and providing an uniform, self-describing API containing all dataset summarizations, in order to improve the discoverability.

As a result of the approach introduced in this paper, we can now provide live queryable access to a large amount of datasets that could previously only be reliably published as data dumps. While it is possible that the current Semantic Web path will eventually lead there, it is worthwhile—and necessary—to explore

alternative stacks already today. Given the solutions it brings to the current Semantic Web problems, we conclude that the technology stack introduced in this paper enables a Semantic Web that is not only technologically, but also economically scalable.

## References


1. Auer, S., Demter, J., Martin, M., Lehmann, J.: LODStats – an extensible framework for high-performance dataset analytics. In: ten Teije, A., Völker, J., Handschuh, S., Stuckenschmidt, H., d'Acquin, M., Nikolov, A., Aussenac-Gilles, N., Hernandez, N. (eds.) EKAW 2012. LNCS, vol. 7603, pp. 353–362. Springer, Heidelberg (2012)
2. Beek, W., Groth, P., Schlobach, S., Hoekstra, R.: A web observatory for the machine processability of structured data on the web. In: Proceedings of the 2014 ACM Conference on Web Science, pp. 249–250. ACM (2014)
3. Beek, W., Rietveld, L., Bazoobandi, H.R., Wielemaker, J., Schlobach, S.: LOD laundromat: a uniform way of publishing other people's dirty data. In: Mika, P., et al. (eds.) ISWC 2014, Part I. LNCS, vol. 8796, pp. 213–228. Springer, Heidelberg (2014)
4. Berners-Lee, T., Hendler, J., Lassila, O., et al.: The semantic web. Sci. Am. **284**(5), 28–37 (2001)
5. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. Int. J. Seman. Web Inf. Syst. **5**(3), 1–22 (2009)
6. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. ACM (2008)
7. Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.-Y.: SPARQL web-querying infrastructure: ready for action? In: Alani, H., et al. (eds.) ISWC 2013, Part II. LNCS, vol. 8219, pp. 277–293. Springer, Heidelberg (2013)
8. Cheng, G., Gong, S., Qu, Y.: An empirical study of vocabulary relatedness and its application to recommender systems. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 98–113. Springer, Heidelberg (2011)
9. Ermilov, I., Martin, M., Lehmann, J., Auer, S.: Linked open data statistics: collection and exploitation. In: Klinov, P., Mouromtsev, D. (eds.) KESW 2013. CCIS, vol. 394, pp. 242–249. Springer, Heidelberg (2013)
10. Feigenbaum, L., Williams, G.T., Clark, K.G., Torres, E.: SPARQL 1.1 protocol. Recommendation, W3C, March 2013. http://www.w3.org/TR/sparql11-protocol/
11. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF representation for publication and exchange (HDT). Web Semant. Sci. Serv. Agents World Wide Web **19**, 22–41 (2013)
12. Ge, W., Chen, J., Hu, W., Qu, Y.: Object link structure in the semantic web. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010, Part II. LNCS, vol. 6089, pp. 257–271. Springer, Heidelberg (2010)
13. Harris, S., Seaborne, A.: SPARQL 1.1 query language. Recommendation, W3C, March 2013. http://www.w3.org/TR/sparql11-query/

14. Haslhofer, B., Isaac, A.: data.europeana.eu: The Europeana linked open data pilot. In: International Conference on Dublin Core and Metadata Applications. pp. 94–104 (2011)
15. Hogan, A., Umbrich, J., Harth, A., Cyganiak, R., Polleres, A., Decker, S.: An empirical survey of linked data conformance. Web Semant. Sci. Serv. Agents World Wide Web **14**, 14–44 (2012)
16. Käfer, T., Abdelrahman, A., Umbrich, J., O'Byrne, P., Hogan, A.: Observing linked data dynamics. In: Cimiano, P., Corcho, O., Presutti, V., Hollink, L., Rudolph, S. (eds.) ESWC 2013. LNCS, vol. 7882, pp. 213–227. Springer, Heidelberg (2013)
17. Linked Data API. https://code.google.com/p/linked-data-api/
18. Matteis, L.: Restpark: Minimal RESTful API for retrieving RDF triples (2013). http://lmatteis.github.io/restpark/restpark.pdf
19. Millard, I., Glaser, H., Salvadores, M., Shadbolt, N.: Consuming multiple linked data sources: challenges and experiences. In: First International Workshop on Consuming Linked Data (COLD 2010), November 2010, Event Dates: 2010-11-07
20. Ogbuji, C.: SPARQL 1.1 Graph Store HTTP Protocol. Recommendation,W3C, March 2013. http://www.w3.org/TR/sparql11-http-rdf-update/
21. Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., Tummarello, G.: Sindice.com: a document-oriented lookup index for open linked data. Int. J. Metadata Semant. Ontol. **3**(1), 37–52 (2008)
22. Saleem, M., Khan, Y., Hasnain, A., Ermilov, I., Ngomo, A.-C.N.: A fine-grained evaluation of SPARQL endpoint federation systems. Seman. Web J. **6**(3) (2015)
23. Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: FedBench: a benchmark suite for federated semantic data query processing. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 585–600. Springer, Heidelberg (2011)
24. Speicher, S., Arwe, J., Malhotra, A.: Linked Data Platform 1.0.Candidate recommendation, W3C, June 2014. http://www.w3.org/TR/2014/CR-ldp-20140619/
25. Verborgh, R., et al.: Querying datasets on the web with high availability. In: Mika, P., et al. (eds.) ISWC 2014, Part I. LNCS, vol. 8796, pp. 180–196. Springer, Heidelberg (2014)
26. Verborgh, R., Vander Sande, M., Colpaert, P., Coppens, S., Mannens, E., Van de Walle, R.: Web-scale querying through linked data fragments. In: Proceedings of the 7th Workshop on Linked Data on the Web, April 2014