How will we interact with the #WebWeWant?
Full paper published at the 5th International USEWOD Workshop: Using the Web in the Age of Data, May 31st, 2015, Portoroz, Slovenia.

Initial Usage Analysis of DBpedia's Triple Pattern Fragments

Ruben Verborgh, Erik Mannens, and Rik Van de Walle
Ghent University – iMinds, Belgium

Abstract

Queryable Linked Data is available through several interfaces, including SPARQL endpoints and Linked Data documents. Recently, the popular DBpedia dataset was made available through a Triple Pattern Fragments interface, which proposes to improve query availability by dividing query execution between clients and servers. In this paper, we present an initial usage analysis of this interface so far. In 4 months time, the server had an availability of 99.999%, handling 4,455,813 requests, more than a quarter of which were served from cache. These numbers provide promising evidence that Triple Pattern Fragments are a viable strategy for live applications on top of public queryable datasets.

Keywords: Linked Data, Linked Data Fragments

Introduction

DBpedia [2] is currently the most well-known dataset within the Semantic Web community. It consists of hundreds of millions of RDF triples automatically generated from the free Wikipedia encyclopedia. Such large Linked Datasets come with important challenges, most prominently: how do we provide scalable queryable access to them? The traditional answer has been to set up a public SPARQL endpoint [4], but such endpoints suffer from low availability rates [3]. Yet reliable access is a prerequisite to build applications on top of a queryable DBpedia interface.

In October 2014, the DBpedia community opened a Triple Pattern Fragments interface [10] maintained by the authors of this paper. This interface is designed to allow high availability on the server side, while still enabling live querying on the client side. Queries take more time and bandwidth, because they are mostly executed by the client, but the timings are consistent so that building applications on top of a public DBpedia interface becomes realistic.

In this paper, we discuss the analysis of 4 months of usage data of the English DBpedia Triple Pattern Fragments interface, as well as availability data measured by an external party (Pingdom).

Related work

In this section, we will briefly discuss existing Web APIs to Linked Datasets. *Linked Data Fragments* (LDF, [10]) were introduced as a uniform view to capture the characteristics of any Linked Data Web API. The common aspect of all interfaces is that, in one way or another, they offer specific parts of a dataset. Each part is referred to as a *Linked Data Fragment*, consisting of:

data the triples of the dataset that match an interface-specific *selector*;

metadata triples to describe the fragment itself;

controls hyperlinks and/or hypermedia forms that lead to other fragments.

File-based datasets

So-called *data dumps* are conceptually the most simple APIs: the *data* consists of all triples in the dataset. They are combined into a (usually 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 is the clients' responsibility.

Linked Data documents

Datasets published through the Linked Data principles [1] are available as individual documents per subject, which can be retrieved by performing an HTTP `GET` request on the subject's URL *("dereferencing")*. Each such document is a fragment, in which the *data* consists of triples related to that subject, the *metadata* set might contain properties such as author and publication data, and the *controls* consist of links to other Linked Data documents. Querying is possible through strategies such as link traversal [7].

SPARQL endpoints

SPARQL endpoints [4] allow executing SPARQL queries [6] on a dataset through HTTP. A SPARQL 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 reusability of fragments is low. This, combined with complexity of SPARQL query execution, likely contributes to the low availability of public SPARQL endpoints [3].

Triple Pattern Fragments

The Triple Pattern Fragments API [10] interface has been designed to minimize server-side 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 mentions 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. Since requests are less individualized, fragments are more likely to be reused across clients, which increases the benefits of caching [10].

Deployment and analysis setup

Server specifications

The official DBpedia Triple Pattern Fragments interface is hosted on a virtual machine from the Amazon Elastic Compute Cloud (EC2). We opted for an c3.2xlarge machine configuration, which has the following characteristics:

virtual CPUs: 8
memory: 15GB
hard disk space: 2 × 80GB
price: $ 0.478 per hour (dedicated instance in Ireland)

We would like to stress that the above specifications are actually too high for our purpose; as a result, the server is currently mostly idle. The issue is, however, that Amazon does not allow customization of machines. While lighter configurations exist, they come with lower disk throughput and/or bandwidth.

The machine runs the following software:

operating system: Ubuntu Linux 14.04 LTS
Web server: nginx 1.4.6
application server: Linked Data Fragments server 1.1.4 on top of Node.js 0.10.36

The nginx server acts as a reverse proxy and cache. All requests first reach nginx, which checks whether a response is present in the cache based on a unique identifier consisting of the request URI and the value of the HTTP `Accept` header. If so, it is sent to the client; if not, the request is forwarded to the application server. The application server then parses the request, and retrieves the DBpedia data from an HDT file [5] that is loaded into memory. It is then serialized in a format according to the `Accept` header, sent to the client, and stored in the cache.

Analysis setup

All incoming requests are logged line by line in a file by the nginx Web server. Note that logging does *not* happen on the application server, as this server only receives those requests that are not handled by the cache. Each log line contains the following fields:

- client IP address
- request URI
- value of the `Accept` header
- value of the `Referer` header
- value of the `User-Agent` header
- local server time
- response size
- response cache status
- response HTTP status code

The resulting access logs are hosted publicly.

Additionally, the availability of the HTTP interface is monitored by the external third-party service Pingdom, because public availability can of course not reliably be monitored by the Web server itself. Pingdom performs an HTTP request once every minute for the `?s rdf:type ?o fragment` and notes whether a response was successfully received. If no timely responses arrives, the server is assumed to be unavailable. The results are available in an online interface.

Usage analysis

In this section, we will search an answer to these basic usage questions:

- How many requests were issued?
- Which clients made these requests?
- What types of content were those clients interested in?
- Where did the requests originate from?
- What kind of triple patterns were requested?
- How effective has the cache been?
- What period of time was the server (un-)available?

We focused on requests with an HTTP `200 OK` response only, in order to remove (very minimal) noise from invalid requests against the interface.

Number of requests

The server logs reveal a total of 4,455,813 requests for Triple Pattern Fragments of the English DBpedia version (URLs starting with `http://fragments.dbpedia.org/2014/en`) during the four considered months, or an average of 1,113,953 requests per month. November 2014 was responsible for twice the average monthly traffic. The majority of this initial traffic originates from a machine within Ghent University (recognizable by the `157.193.0.0/16` IP address block), which was used to stress test the new server and measure the execution times of various simple and complex queries. In the other months, the traffic was much more varied.

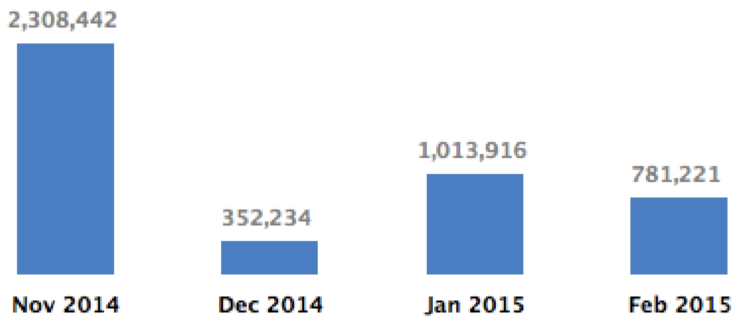

Figure 1: November saw the highest usage of fragments, mostly due to stress testing by us. The average is about a million requests per month.

User agents

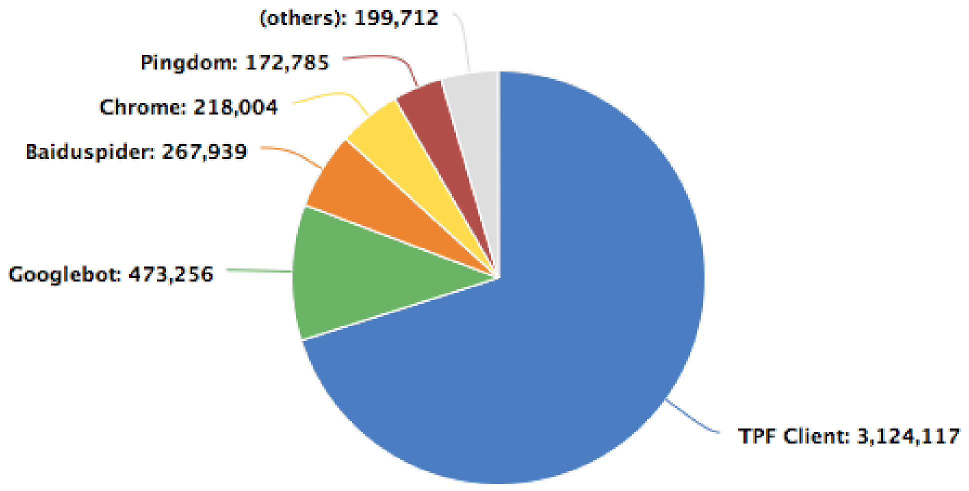

Figure 2: The dedicated Triple Pattern Fragments client consumed most fragments, followed by crawlers of search engines.

By extracting and parsing the value of the HTTP `User-Agent` header sent by clients, we were able to see what kinds of clients were interested in DBpedia's Triple Pattern Fragments. The vast majority of requests (70.1%) were performed by the Node.js Triple Pattern Fragments client, which executes SPARQL queries by requesting triple patterns. This client can either be used in a standalone manner, or as a library for other applications. We cannot distinguish between access made by the standalone client and access made by other software packages that use the client as a library. To mitigate this in the future, we should suggest that such other software packages use their own user agent identifier.

The second and third most active clients were crawlers from the search engines Google and Baidu respectively. This is especially remarkable because it contrasts with SPARQL endpoints, which belong to the so-called "deep Web" [8]: in order to access data, a user must write a SPARQL query in an HTML form. The only SPARQL endpoint resources that are accessible on the Web are SPARQL queries that are explicitly linked from another page (such as this one). While the Triple Pattern Fragments specification only demands the presence of a hypermedia form (which would thus also hide fragments in the deep Web), the server implementation explicitly links to relevant fragments. For instance, the subjects born in Slovenia fragment links to fragments for the birthplace predicate, Slovenia, and all individual subjects born in Slovenia. This allows people and crawlers to browse the interface similar to how Linked Data documents are navigated. An added value of Triple Pattern Fragments is that *all* resources can be followed within the interface, not only those resources that share the URI space of the current document (as is the cases with Linked Data documents).

The statistics also reveal browser usage, mainly through the Chrome browser. The `Accept` header tells us that, out of 218,004 requests by various Chrome versions, 216,828 were performed by the in-browser version of the Triple Pattern Fragments client during the execution of SPARQL queries; 782 requests consist of HTML pages viewed by humans.

Note the 172,785 requests performed by the Pingdom bot to monitor the availability of the interface. Not shown in the graph are 53,848 requests from `Apache-HttpAsyncClient`, which likely originate from the early-stage Java Triple Pattern Fragments client. Finally, the Perl client deserves a honorable mention with 19,287 requests.

Requested content types

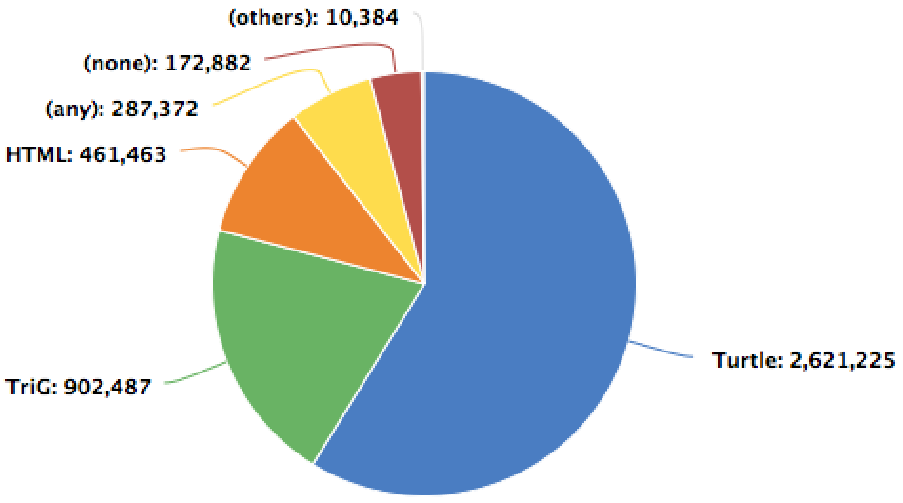

Figure 3: Turtle was the most popular content type, but will likely be surpassed by TriG. Browsers and some crawlers prefer HTML.

The Triple Pattern Fragments interface exposes the same fragments through the same URLs, regardless of content type. This is achieved through HTTP content negotiation. For instance, the fragment "subjects born in Slovenia" has the URL `http://fragments.dbpedia.org/2014/en?predicate=http%3A%2F%2Fdbpedia.org%2Fontology%2FbirthPlace&object=http%3A%2F%2Fdbpedia.org%2Fresource%2FSlovenia.` In order to retrieve an HTML representation, a client should send an HTTP `GET` request with an `Accept` header that prefers HTML; the same goes for other content types such as Turtle or JSON. We analyzed the requested representations by looking at clients' most preferred options.

Since each type of client usually consumes a specific format, the distribution of clients strongly influences the requested content types. It is therefore expected that the content type requested by the Triple Pattern Fragments client (both standalone and in-browser) prevails. We indeed see that the majority of requests (58.8%) has a preference for Turtle, which used to be this client's preferred format up to version 1.2.1. From version 1.2.2 onwards, support for the quad-based serialization format TriG was added, which uses graphs to separate fragment data from metadata and controls. Hence, we also see a large amount (20.3%) of TriG requests—and this is expected to dominate in the future as older client versions disappear.

Browsers of course prefer HTML variants for displaying to humans. Some clients (mostly crawlers) indicated they had no specific preference (`*/*`). The Pingdom bot does not indicate any `Accept` header and is the major contributor to the *none* category. Other requested formats include JSON (for which JSON-LD representations were returned) and other RDF formats such as N-Quads and N-Triples.

Geographic location

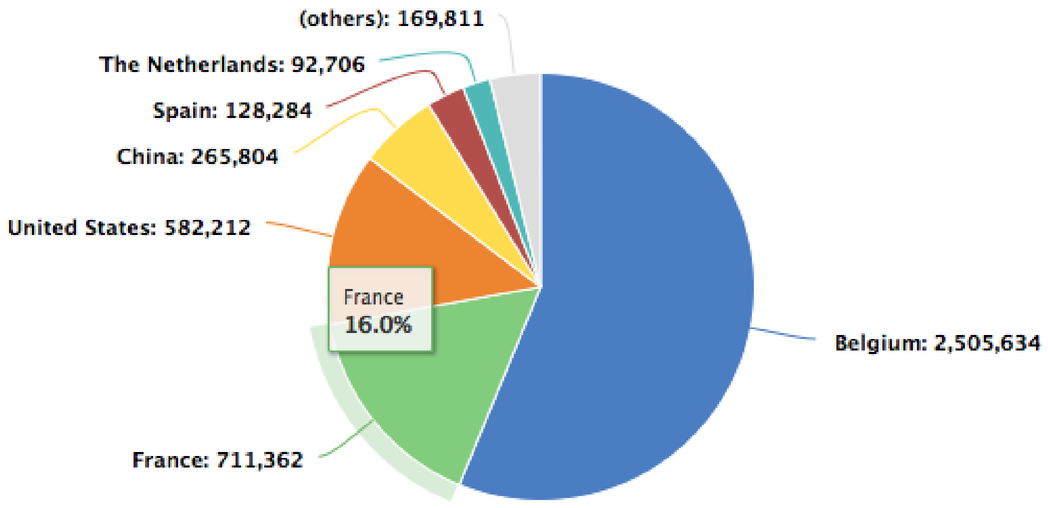

Figure 4: The majority of requests originated from Belgium due to our intensive tests running in the beginning period. Traffic from other countries is growing.

In order to determine the geographic origins of requests, we performed automated lookups on the client IP addresses. As indicated earlier, stress tests by Ghent University were performed during the interface's first weeks. This is visible in the large portion originating from Belgium (56.2%). 188,891 Belgian requests (4.24% of all requests) did *not* originate from within Ghent University. After France, clients from the United States and China were popular visitors, mostly through search engine crawlers.

In total, the interface received traffic from 47 countries, 17 of which sent at least 1,000 requests.

Requested triple patterns

If an interface allows highly specific queries, like SPARQL endpoints do, we expect a great variety of requests on the server side. Also, this brings detailed insights in the kind of goals clients have. Since the Triple Pattern Fragments interface is deliberately more simple, we expect to see more repeated queries—but less insights in how these smaller queries contribute to a goal for the client.

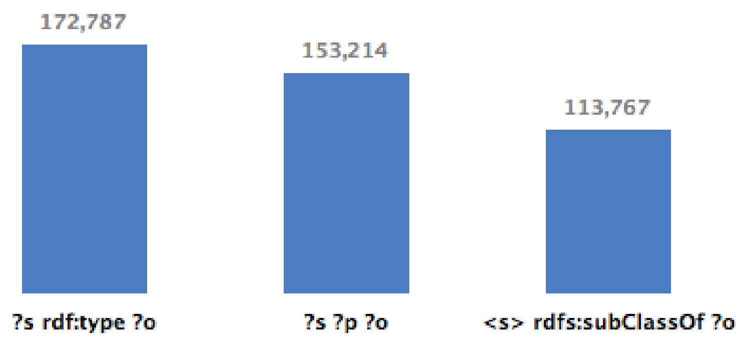

Figure 5: The fragment for *things that have a type* was requested most (by Pingdom), followed by the generic *all* fragment (because it is often used to bootstrap the query process).

Unsurprisingly, the fragment requested every minute by Pingdom (`?s rdf:type ?o`) as part of its availability monitoring process is most popular. This is closely followed by the *all* fragment. Since this is the most generic fragment of the dataset, clients in practice often use it to start their more complex process; i.e., it is the first form they fill out. However, *any* fragment can in theory be used as a starting point, as the Triple Pattern Fragments specification requires all of them to contain the same hypermedia controls. Since the *all* fragment is a straightforward starting point, the 153,214 requests to this fragment format give a vague indication of the total number of SPARQL queries that were executed by clients. As clients might also perform other tasks, this number is likely inaccurate (and as more clients are developed, this will become only more vague). For instance, the `Referer` header values reveal that 8,955 requests originated from the UDUVUDU DBpedia Viewer, which visualizes topics from DBpedia. Finally, we note a high number of `<s> rdfs:subClassOf ?o` requests for specific instances of `<s>`. They were caused by the stress testing queries we issued, which contained `rdfs:subClassOf` constructs.

Other than the three cases above, no obvious patterns were found in the requests.

Cache effectiveness

A premise of the Triple Pattern Fragments interface is that clients partly reuse the same fragments to achieve different but similar goals. With SPARQL endpoints, clients instead send highly specialized requests; overlapping information between them cannot be reused on the HTTP interface level. With Triple Pattern Fragments, the number of *unique* requests is relatively smaller, so the cache can work more effectively.

The nginx reverse proxy server has been configured to cache requested fragments for a maximum time of 1 hour. Uniqueness of requests is determined by a combination of URL and `Accept` header. As such, the Triple Pattern Fragments server generates each unique response at most once per hour; all subsequent requests are handled by the cache. Furthermore, the proxy server sets the expiration date of responses to 7 days in the future. Clients that have a built-in cache themselves, such as browsers, are thereby suggested to only repeat a request for a resource after a week. Note that the standalone client does not have a persistent cache; therefore, each invocation of that client results in new resource accesses.

In total, 28.1% of responses were served from the nginx cache. This means that between a quarter and a third of all responses were needed again by the same client or other clients within the hour. A minority of 3.5% had been present in the cache for longer than 1 hour, so new versions needed to be fetched. Finally, a few requests (1,278) explicitly asked to bypass the cache. So while the majority of requests was not cached, the caching mechanism was able to reduce the load on the application server by 28.1%. Since the dataset in this case is static, and the number of fragments finite, we could set a higher (of even infinite) cache timeout. At the moment, however, there was no necessity to do so.

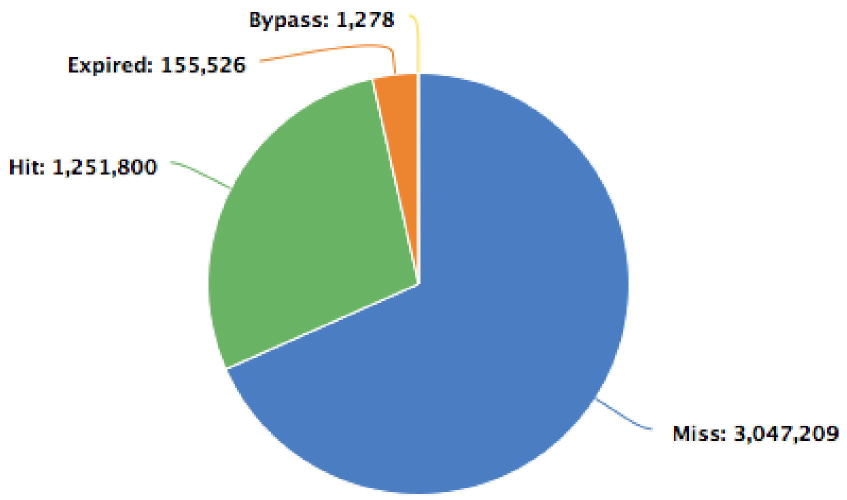

Figure 6: A quarter of responses was cached. In theory, all content could be pre-cached, since the number of Triple Pattern Fragments per dataset is finite.

Availability

One of the main goals of the Triple Pattern Fragments interface is to maximize availability, in order to allow building applications on public, live queryable Linked Data sources. During the period of November 2014 to February 2015, a fragment was retrieved from the server every minute to verify availability. This amounts to a total of 120 days × 1,440 minutes per day = 172,800 minutes. According to Pingdom, only 1 of those requests did not receive a timely answer; this occurred on 2014/11/26 at 3:52pm CET. What exactly happened at this moment is unclear; nothing particular shows up in the server logs at this point, except an unexpected gap between two Pingdom requests at 14:50 and 14:52 UTC (server time). We presume it is an Amazon-related outage due to lack of evidence of software malfunction at this moment in time.

In any case, the above allows to precisely calculate the availability during the observed period of 4 months. Dividing the minutes of availability by the total number of minutes gives 172,799 / 172,800 = 99.99942…%. This amounts to an availability level of "5 nines", or on average maximum 25 seconds downtime per month (assuming months of 30 days).

Note that the total number of requests logged from Pingdom (172,785 as indicated above) is 15 short of the expected total of 172,800. Since Pingdom did not report any other outages, we are unsure about the cause. Slightly incomplete logging could be a straightforward explanation, for instance, if Pingdom dropped the connection before the full response was received.

Conclusions

When the official Triple Pattern Fragments interface for DBpedia was released, we mostly heard three types of questions:

1. Will this interface be used?
2. If so, how will clients use it?
3. Will the availability of this interface be sufficient for live application usage?

The analysis in this paper allows us to formulate a preliminary answer on all three of them.

First of all, the interface has indeed been used, as evidenced by more than 4 million requests in the course of its first 4 months. Most of this usage came from the client-side SPARQL query executor we previously built for the Triple Pattern Fragments interface, but we also saw third-party clients such as a Perl client and a DBpedia viewer. Search engine crawlers also consumed the interface with ease. Relatively few people browsed the interface directly, as it is of course targeted at machines. It does raise the question whether it makes sense to improve accessibility for people. Client IP addresses from 47 countries show that usage is spreading geographically.

Second, while the analysis provides us with some insights about how the interface is used, more high-level patterns are absent. On the one hand, this is a blessing for privacy: clients only ask generic questions, and they themselves can combine this to answers for more complex questions in any way they see fit. On the other hand, it makes it harder to understand what kind of usage is popular, and for which use cases we could or might need to optimize. This process could be facilitated if we explicitly ask clients to provide feedback [9]. For now, we are in the dark as to precisely what SPARQL queries—and other tasks—clients have executed. Having more information would allow us to compare this with, for instance, the logs of the public DBpedia SPARQL endpoint. At the same time, we should realize that not all clients of Triple Pattern Fragments interfaces necessarily have the evaluation of SPARQL queries as a task or subtask.

Third, the 99.999% availability of the server removes any doubt that the Triple Pattern Fragments interface is sufficiently reliable for live applications. We must, however, remark two things here. While 4 million requests is a large quantity for a young interface, it is still nowhere near full capacity. The server is still mostly idling, so in order to really find out its limits, more requests are necessary. Also, the number of requests cannot be compared to that of a SPARQL endpoint, as in many cases, more requests are necessary to achieve the same goal. When talking about availability, we therefore need to mention expressivity too. The goal of the Triple Pattern Fragments interface is to reliably balance both.

Our conclusion is that applications now have a reliable interface to query the public DBpedia dataset. Therefore, we seem to have overcome one of the main obstacles that could hold developers from building applications on top of live Linked Data. An important question remains: *is this enough?* Now that reliable access is possible, what excuses remain for not building intelligent Linked Data clients? It seems the next move should be made by application developers, given that the data and the tools are now *really* there, 99.999% of time. We should keep our eyes, ears, and minds open to the demands of this community to help evolve the concept of *Semantic Web applications* from vision to reality.

Acknowledgements

The described research activities were funded by Ghent University, iMinds, the Institute for the Promotion of Innovation by Science and Technology in Flanders, the Fund for Scientific Research Flanders, and the European Union.

Pingdom graciously provided us with availability monitoring. The geographic analysis was performed using GeoLite data created by MaxMind. Special thanks to Dimitris Kontokostas from the DBpedia Association for giving us the opportunity to host DBpedia as Triple Pattern Fragments.

References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data – the story so far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (Mar 2009)
2. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: DBpedia – a crystallization point for the web of data. Journal of Web Semantics 7(3), 154–165 (2009)
3. Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.Y.: SPARQL Web-querying infrastructure: Ready for action? In: Proceedings of the 12th International Semantic Web Conference (Nov 2013)
4. Feigenbaum, L., Williams, G.T., Clark, K.G., Torres, E.: SPARQL 1.1 protocol. Recommendation, World Wide Web Consortium (Mar 2013)
5. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF representation for publication and exchange (HDT). Journal of Web Semantics 19, 22–41 (Mar 2013)
6. Harris, S., Seaborne, A.: SPARQL 1.1 query language. Recommendation, World Wide Web Consortium (Mar 2013)
7. Hartig, O.: An overview on execution strategies for Linked Data queries. Datenbank-Spektrum 13(2), 89–99 (2013)
8. Madhavan, J., Ko, D., Kot, Ł, Ganapathy, V., Rasmussen, A., Halevy, A.: Google's Deep Web crawl Proceedings of the VLDB Endowment 1(2), pp. 1241–1252 (Aug 2008)
9. Verborgh, R.: The Lonesome LOD Cloud In: Proceedings of the Fourth Workshop on Usage Analysis and the Web of Data (May 2014)
10. Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., Vander Sande, M., Cyganiak, R., Colpaert, P., Mannens, E., Van de Walle, R.: Querying datasets on the Web with high availability. In: Proceedings of the International Semantic Web Conference. Lecture Notes in Computer Science, vol. 8796, pp. 180–196 (Oct 2014)