DBpedia's Triple Pattern Fragments: Usage Patterns and Insights

Ruben Verborgh

Ghent University – iMinds, Belgium
ruben.verborgh@ugent.be

Abstract. Queryable Linked Data is published through several interfaces, including SPARQL endpoints and Linked Data documents. In October 2014, the DBpedia Association announced an official Triple Pattern Fragments interface to its popular DBpedia dataset. This interface proposes to improve the availability of live queryable data by dividing query execution between clients and servers. In this paper, we present a usage analysis between November 2014 and July 2015. In 9 months time, the interface had an average availability of 99.99%, handling 16,776,170 requests, 43.0% of which were served from cache. These numbers provide promising evidence that low-cost Triple Pattern Fragments interfaces provide a viable strategy for live applications on top of public, queryable datasets.

Keywords: Linked Data, Linked Data Fragments, DBpedia

1 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.

Mid-October 2014, the DBpedia community opened a Triple Pattern Fragments interface[1] [14] maintained by the author 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 the first 9 full months of usage data of the English DBpedia Triple Pattern Fragments interface, as well as availability statistics measured by an independent party.

[1] Available at `http://fragments.dbpedia.org/2014/en`.

The remainder of this paper is structured as follows. First, we discuss interfaces to Linked Data in Section 2. We then discuss the hardware and software setup of the server and analysis in Section 3. Next, Section 4 formulates and answers usage questions with log data. Finally, we conclude in Section 5.

2 Related Work

In this section, we summarize existing Web APIs to publish Linked Datasets. *Linked Data Fragments* (LDF, [14, 16]) 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.

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 interface has been designed to minimize server-side processing, while at the same time enabling efficient live querying on the client side [11, 14]. 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 as metadata, 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 [14].

3 Deployment and Analysis Setup

3.1 Server Specifications

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

virtual CPUs	8 (Intel Xeon E5-2680 v2)
memory	15 GB
hard disk space	2 × 80 GB
price	$0.478 per hour

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.

Software The machine has been configured with the following open-source software packages and versions:

operating system	Ubuntu Linux 14.04 LTS
Web server	nginx 1.4.6
application server	LDF server[2] 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.

3.2 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. The resulting access logs are hosted publicly.[3] Each log line contains the following fields:

1. the client's IP address;
2. the URI requested by the client;
3. value of the client's `Accept` header;
4. value of the client's `Referer` header;
5. value of the client's `User-Agent` header;
6. the server's local time;
7. the server's response size;
8. the server's response cache status;
9. the server's response HTTP status code.

[3] Available at `http://fragments.dbpedia.org/logs/`.

Additionally, the availability of the HTTP interface is monitored by the independent third-party service Pingdom, because public availability—by definition—cannot reliably be monitored by the Web server under consideration. 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 response arrives, the interface is assumed to be unavailable.

4 Usage Analysis

In this section, we will search for answers to the following usage questions:

1. How many requests were issued?
2. Which clients made these requests?
3. What types of content were those clients interested in?
4. Where did the requests originate from?
5. What kind of triple patterns were requested?
6. How effective has the cache been?
7. What part 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 the 0.19% invalid requests against the interface.

4.1 Number of Requests

The server logs reveal a total of 16,776,170 requests for Triple Pattern Fragments of the English DBpedia version[4], or an average of 1,864,019 requests per month during the 9 considered months (November 2014–July 2015). Large outliers skew this average, so perhaps the median of 486,045 (obtained in March) is a more meaningful number. Fig. 1 visualizes the number of requests per month.

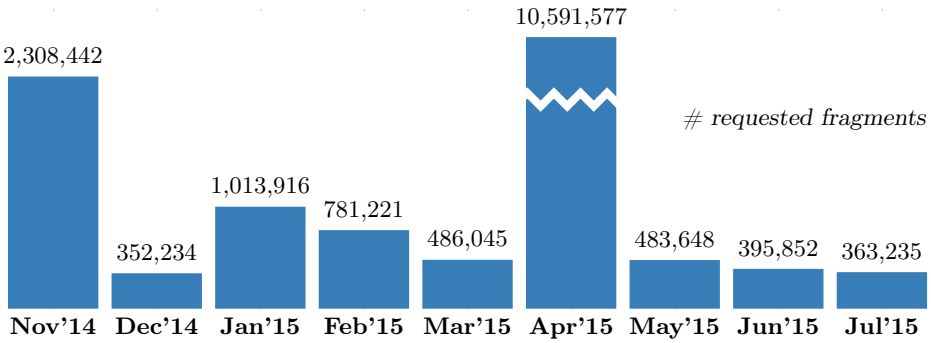

Fig. 1. April had an exceptionally high consumption of fragments, followed by November. The median is 486,045, which was obtained in March.

[4] URLs starting with `http://fragments.dbpedia.org/2014/en`.

April 2015 had a strong traffic peak with over 10 million requests. While we have no concrete evidence, we assume there is a connection with the submission deadline of the International Semantic Web Conference 2015: main track papers were due April 30, 2015, so perhaps one or more research groups were running experiments against the interface.

4.2 User Agents

Fig. 2 shows the proportion of user agents per requested fragment. The majority of requests were issued by clients that identified themselves as Triple Pattern Fragment (TPF) clients. This is the identification sent by the JavaScript client-side library,[5] which can either be used as standalone utility from the command-line or as a library inside of other applications (which thus cannot be distinguished by default). If the library runs inside of a browser application, the user agent will be that of the browser, so these usages are counted in that category.

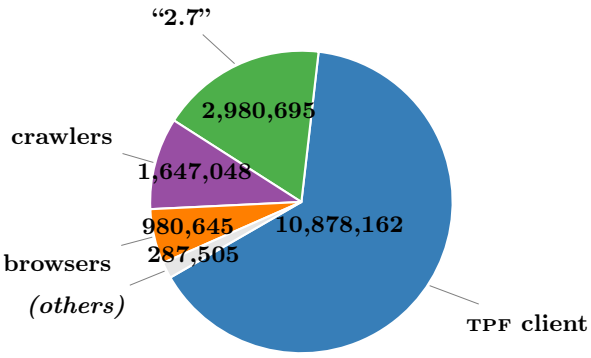

Fig. 2. The Triple Pattern Fragments client is by far the most common client, followed by an unknown client, crawlers, and finally browsers. Some minorities (command-line utilities, other languages...) are marginally represented.

The second-most popular client identifies itself with "`User-Agent: 2.7`", which requested approximately 3 million requests, all of which originated from Germany. This seems to be an error, as "`2.7`" does not follow a conventional format for user agent strings. We could not determine the identify of this client more precisely. There might be a connection with Python 2.7 (which is a common version), but other Python clients identified themselves in a more conventional way (e.g., "`python-requests/2.7.0`").

Crawlers requested a large portion of DBpedia fragments. This is especially remarkable because such usage contrasts with SPARQL endpoints, which belong to the so-called "deep Web": in order to access data, a user must write a SPARQL

[5] Source code available at `https://github.com/LinkedDataFragments/Client.js`

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. 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 used server implementation explicitly links to relevant fragments. For instance, the fragment "subjects born in Slovenia" links to fragments for the birthplace predicate, 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 [16]).

Finally, browser consumption accounted for almost a million requests, the majority of which (840,006) appear to be performed by client-side scripts. This latter number was determined by looking for non-human-targeted content types such as Turtle or JSON, which we discuss in more detail in the next section.

4.3 Requested Content Types

Fig. 3 shows the preferred content type indicated in the `Accept` header of the request. Given that the TPF client made most of the requests (Fig. 2), this client's preferred content type TriG is also the most popular. Previous versions of the TPF client consumed Turtle, which partly accounts for the popularity of this format in Fig. 3. The drawback of triple-based formats such as Turtle is that they do not allow a clean separation of data and metadata, while quad-based formats like TriG do [14]. In a previous analysis, we predicted the increased popularity of TriG because of this changed client preference [15]. The popularity of JSON(-LD) might come as a surprise for the Linked Data community, but perhaps less so for JavaScript developers, who employ JSON as a native format.

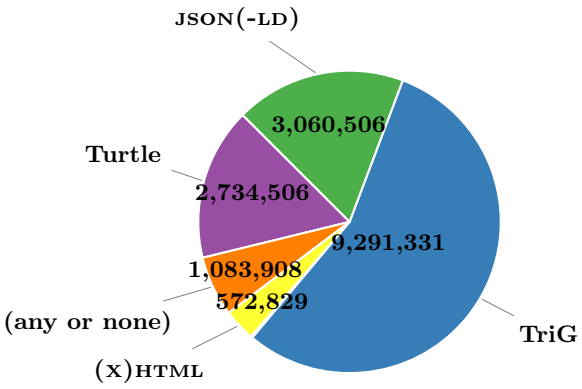

Fig. 3. The quad-based TriG and JSON-LD formats were most popular, followed by the triple-based format Turtle. Over a million requests did not carry a preference.

Only a minority of clients did not indicate a specific preference, either by explicitly accepting `*/*` or not sending an `Accept` header at all. These clients received an HTML representation from the server.

4.4 Geographic Location

The majority of requests originated from Germany, as visualized in Fig. 4. On the second place is Belgium; most Belgian requests were generated by our team at Ghent University – iMinds, which coordinates the DBpedia Triple Pattern Fragments server. Requests from the United States and China were largely made by respectively the Google and Baidu crawlers.

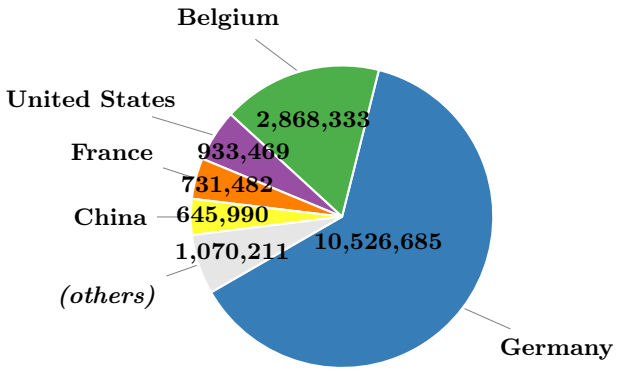

Fig. 4. Germany requested most fragments, followed by Belgium. Automated crawlers strongly influence consumption from the United States (Google) and China (Baidu).

In total, requests arrived from 80 different countries. While determining more detailed locations is possible (e.g., city level), we decided not to do so out of privacy concerns.

4.5 Requested Triple Patterns

SPARQL endpoints typically receive highly specific queries that provide an insight into a concrete question. If Triple Pattern Fragments interfaces are used to evaluate SPARQL queries, only triple patterns arrive at the server. Such individual patterns only provide a limited idea of what clients were doing, which could be enhanced by analyzing series of requests made by particular clients. However, there is no guarantee that clients are actually executing SPARQL queries (except perhaps if the user agent is "`TPF Client`"), since the interface can be used in many different ways. For example, crawlers do not evaluate complex queries.

Pre-print of a paper accepted to the ESWC 2015 Workshops Post Proceedings.
The final publication is available at link.springer.com.

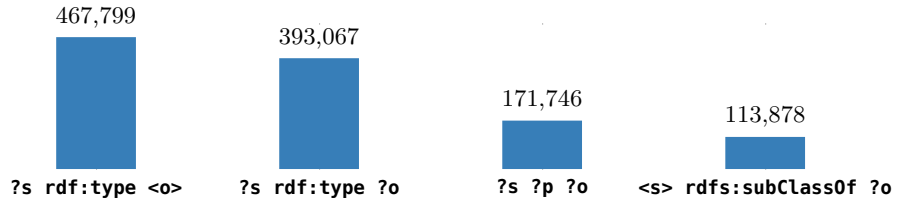

Fig. 5. Type and subclass selections were popular, as well as the generic "all" fragment.

Fig. 5 shows the four most requested kinds of patterns:

1. *"Subjects of a specific type"* fragments (`?s rdf:type <o>` for specific `<o>`) are most common. We expect such patterns to occur frequently in SPARQL queries.
2. The pattern *"subjects that have a type"* (`?s rdf:type ?o`) is highly popular. The main reason for this is that the Pingdom service (see Section 4.7) requests this fragment to measure availability.
3. The *"all"* fragment (`?s ?p ?o`), which is the most generic fragment of the dataset and thus a natural starting point, was requested 171,746 times. Query evaluations typically start from this fragment (but this is by no means an obligation). This number might thus be a vague indication for the number of executed SPARQL queries.
4. *"Subclasses of a specific subject"* fragments (`<s> rdf:type ?o` for specific `<s>`) were also represented significantly in the total number.

The information these patterns yield is quite limited. While this is positive for privacy on the one hand, it restricts the possibilities for analysis on the other hand. In Section 5, we review possible strategies to circumvent this obstacle.

4.6 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 regular HTTP caches function 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 TPF client does not have a persistent cache; each invocation of that client results in new resource accesses.

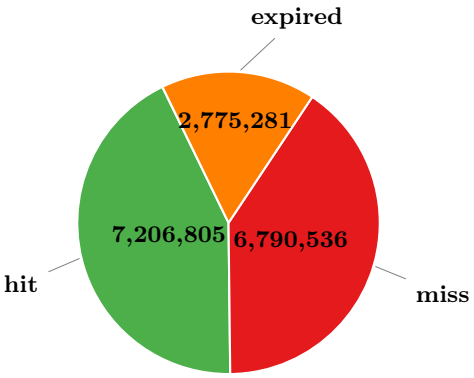

Fig. 6. 59.5% of requested fragments was already present in the cache; 16.5% had expired, but 43.0% could be reused. The remaining 40.5% was not cached.

As Fig. 6 shows, 43.0% of responses were served directly from the nginx cache; another 16.5% were present but had been so for longer than the expiration time. In other words, roughly two fifths of all responses were needed again by the same client or other clients within the hour. Finally, a few requests (3,054) explicitly asked to bypass the cache. So while 57.0% of requests were not served by the cache, the caching mechanism was able to reduce the load on the application server by 43.0%. 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.

4.7 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 July 2015, a fragment was retrieved from the server every minute to verify availability. This amounts to a total of 273 days × 1,440 minutes per day = 393,120 minutes. The results are available in an online interface[6] and summarized in Fig. 7.

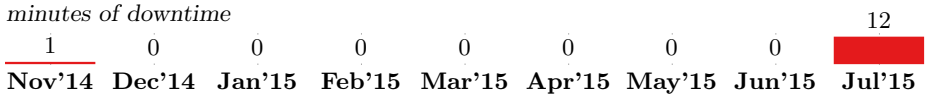

Fig. 7. Most months had 100% uptime. The total downtime was limited to 13 minutes in the 9 full months the interface has been available so far.

[6] Available at `http://stats.pingdom.com/tpb64v451f9p/1382520`.

Pre-print of a paper accepted to the ESWC 2015 Workshops Post Proceedings.
The final publication is available at link.springer.com.

The results reveal that the interface had 100% uptime in 7 months out of 9. There were three separate downtime incidents during which ping requests did not result in a timely response: one in November (1 minute) and two in July (1 minute and 11 minutes). In all of these cases, the Pingdom requests did not even reach the nginx server, as evidenced by a lack of log entries during those periods. There is no indication that the application server was unavailable or overloaded. The cause of the incidents could unfortunately not be determined.

In any case, these observations allow a precise calculation of the availability during the observed period of 9 months. Dividing the minutes of availability by the total number of minutes gives $393{,}107/393{,}120 \approx 99.9967\%$. This amounts to an availability level of "4 nines", which can only be achieved with an average maximum of 4 minutes of downtime per month, which was met in all months except July. The next availability level ("5 nines") would restrict downtime to 26 seconds per month, which was met in all months except November and July.

5 Conclusions

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

- Will this interface be used?
- If so, how will clients use it?
- 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 16 million requests in the course of its first 9 months. Most of this usage came from the client-side SPARQL query executor we built for the Triple Pattern Fragments interface, while can serve as a library for many types of applications. Search engine crawlers also consumed many fragments of the interface. Relatively few people browsed the interface directly, as it is primarily targeted at machines. It does raise the question whether it makes sense to improve accessibility for people. Client IP addresses from 80 countries (as opposed to 47 in a previous analysis [15]) 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 [13]. 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.

Pre-print of a paper accepted to the ESWC 2015 Workshops Post Proceedings.
The final publication is available at link.springer.com.

Third, the average availability of 99.99% of the server (100% in 7 months out of 9) removes any doubt that the Triple Pattern Fragments interface is sufficiently reliable for live applications. We must, however, remark two things here. While 16.8 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.

In the future, we should consider experimenting with more expressive interfaces. For instance, we could provide extra functionality such as substring searches [10], which enable faster evaluation of SPARQL queries with certain `FILTER` clauses. It could also drive specific tools and applications such as autocompletion widgets or linking and reconciliation tasks [8]. To improve query performance, the incorporation of additional metadata in responses can be beneficial [12].

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, 99.99% 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. Furthermore, the DBpedia use case can act as an inspiration for others who want to publish queryable Linked Data at low cost—or even for free [9].

Acknowledgements

Ruben Verborgh is a Postdoctoral Fellow of the Research Foundation Flanders.

Pingdom (`https://www.pingdom.com/`) 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), `http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf`
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), `http://www.sciencedirect.com/science/article/pii/S1570826809000225`

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), http://link.springer.com/chapter/10.1007/978-3-642-41338-4_18
4. Feigenbaum, L., Williams, G.T., Clark, K.G., Torres, E.: SPARQL 1.1 protocol. Recommendation, World Wide Web Consortium (Mar 2013), http://www.w3.org/TR/sparql11-protocol/
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), http://www.w3.org/TR/sparql11-query/
7. Hartig, O.: An overview on execution strategies for Linked Data queries. Datenbank-Spektrum 13(2), 89–99 (2013), http://dx.doi.org/10.1007/s13222-013-0122-1
8. van Hooland, S., Verborgh, R., De Wilde, M., Hercher, J., Mannens, E., Van de Walle, R.: Evaluating the success of vocabulary reconciliation for cultural heritage collections. Journal of the American Society for Information Science and Technology 64(3), 464–479 (2013), http://freeyourmetadata.org/publications/freeyourmetadata.pdf
9. Matteis, L., Verborgh, R.: Hosting queryable and highly available Linked Data for free. In: Proceedings of the ISWC Developers Workshop 2014 (Oct 2014), http://ceur-ws.org/Vol-1268/paper3.pdf
10. Van Herwegen, J., De Vocht, L., Verborgh, R., Mannens, E., Van de Walle, R.: Substring filtering for low-cost Linked Data interfaces. In: Proceedings of the 14th International Semantic Web Conference (Oct 2015), http://linkeddatafragments.org/publications/iswc2015-substring.pdf
11. Van Herwegen, J., Verborgh, R., Mannens, E., Van de Walle, R.: Query execution optimization for clients of Triple Pattern Fragments. In: Proceedings of the 12th Extended Semantic Web Conference (Jun 2015), http://linkeddatafragments.org/publications/eswc2015.pdf
12. Vander Sande, M., Verborgh, R., Van Herwegen, J., Mannens, E., Van de Walle, R.: Opportunistic Linked Data querying through approximate membership metadata. In: Proceedings of the 14th International Semantic Web Conference (Oct 2015), http://linkeddatafragments.org/publications/iswc2015-amf.pdf
13. Verborgh, R.: The lonesome LOD cloud. In: Proceedings of the 4th USEWOD Workshop on Usage Analysis and the Web of Data (May 2014), http://people.cs.kuleuven.be/~bettina.berendt/USEWOD2014/verborgh_usewod2014.pdf
14. 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 (2014), http://linkeddatafragments.org/publications/iswc2014.pdf
15. Verborgh, R., Mannens, E., Van de Walle, R.: Initial usage analysis of DBpedia's Triple Pattern Fragments. In: Proceedings of the 5th USEWOD Workshop on Usage Analysis and the Web of Data (Jun 2015), http://linkeddatafragments.org/publications/usewod2015.pdf
16. 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 (Apr 2014), http://events.linkeddata.org/ldow2014/papers/ldow2014_paper_04.pdf