Hosting Queryable and Highly Available Linked Data for Free

Luca Matteis[1] and Ruben Verborgh[2]

[1] Bioversity International
Via dei Tre Denari 472/a
00057 Maccarese (Fiumicino) Rome, Italy
l.matteis@cgiar.org
[2] Multimedia Lab — Ghent University — iMinds
Gaston Crommenlaan 8 bus 201
B-9050 Ledeberg-Ghent, Belgium
ruben.verborgh@ugent.be

Abstract. SPARQL endpoints suffer from low availability, and require to buy and configure complex servers to host them. With the advent of Linked Data Fragments, and more specifically Triple Pattern Fragments (TPFs), we can now perform complex queries on low-cost servers. Online file repositories and cloud hosting services, such as GitHub, Google Code, Google App Engine or Dropbox can be exploited to host this type of linked data for free. For this purpose we have developed two different proof-of-concept tools that can be used to publish TPFs on GitHub and Google App Engine. A generic TPF client can then be used to perform SPARQL queries on the freely hosted TPF servers.

Keywords: Linked Data, hosting, querying, availability

1 Introduction

Linked Open Data incorporates a large amount of structured data on the Web [1]. The availability of all this interlinked data presents the possibility to query this emerging dataspace as if it was a huge multidatabase system. Linked datasets are often published in a variety of different ways, such as RDF dumps, subject pages or SPARQL endpoints [2]. RDF dumps can be downloaded and processed locally. Subject pages can be crawled and dynamically dereferenced [3]. Finally SPARQL endpoints can be queried online without having to download the entire linked dataset [4].

Although SPARQL endpoints satisfy the need for live querying linked datasets, they suffer from low availability and require users to buy, manage and configure complex servers to host them [5]. Linked Data Fragments offer alternatives to SPARQL for publishing queryable linked data. Specifically, Triple Pattern Fragments (also known as *basic* Linked Data Fragments), a type of Linked Data Fragments, enable users to host linked data on low-cost servers with higher availability than SPARQL endpoints, and allow intelligent agents to perform complex queries [6].

Various free online file hosting services and cloud computing platforms can be utilized to host TPFs. We have developed two applications that enable users to host TPFs on GitHub Pages and Google App Engine, called LDstatic and LDF-GAE respectively. LDstatic allows users to effectively transform RDF files into linked data resources which can then be published on low-cost servers, such as static Web servers. LDF-GAE on the other hand uses Google's cloud computing platform to implement a TPF server on top of their high-replication datastore. We will next describe the functionalities and limitations of both tools.

2 LDstatic

LDstatic[3] allows users to convert RDF graphs into static files, each representing a possible triple pattern of the original RDF source, which can then be published on file hosting services as TPFs. GitHub Pages[4], for example, can host static resources such as HTML files, often used for the creation of static websites, but can also serve other types of content such as N-Triples.

GitHub Pages allows for content to be served via HTTP and returns the appropriate content type based on the file extension of the requested resource. Specifically we can host N-Triples files with `.nt` extensions, which will be served using the `text/plain` content type header. We chose N-Triples because GitHub Pages does not support the Turtle mime type, however, other services might support it. GitHub Pages therefore acts as a static HTTP Web server where various linked data resources can be published.

2.1 Generating Static Resources from RDF Files

The Triple Pattern Fragments specification [7] states that for each triple pattern selector { `?s ?p ?o.` } the returned data must contain triples that match the selector. Working with static resources means that for each possible pattern a file needs to be generated. Maximum 8 possible selector patterns can be performed on a single triple statement. Thus 8 separate files are needed to fulfill a triple pattern request. Each file has a name of a possible pattern and content matching the requested selector. The file hosting provider then simply serves the file based on the file name.

We used Apache Jena[5] and more specifically the StreamRDF API to efficiently parse the RDF data sources. StreamRDF is performant and uses little memory as it does not have to store the entire RDF data in memory. We chose Apache Jena because of its streaming capabilities but also because of its ability to parse a large variety of RDF formats.

[3] `https://github.com/lmatteis/ldstatic`
[4] `https://pages.github.com/`
[5] `https://jena.apache.org/`

2.2 Hashing Triple Nodes

As mentioned earlier, the triple pattern selector is effectively the file name. Having HTTP URIs in the file name can be problematic, as operating systems and file systems have different limitations regarding the maximum length of a file name [8].

For this reason all the nodes of a triple pattern selector are hashed. The advantage of a hashed pattern is that its size is always predictable. As an example, we show a common triple pattern selector, compared to a hashed version, and finally our generated file name:

Common triple pattern selector

```
?predicate= ↩
 http%3A%2F%2Fwww.w3.org%2F1999%2F02%2F22-rdf-syntax-ns%23type ↩
 &object=http%3A%2F%2Fdbpedia.org%2Fontology%2FArtist
```

Hashed triple pattern selector

```
?predicate= ↩
 3c197cb1f6842dc41aa48dc8b9032284bcf39a27 ↩
 &object=2ffecb18d7dac5b51c1159445007956f095a4f2c
```

Generated file name

```
-P-3c197cb1f6842dc41aa48dc8b9032284bcf39a27-O-2ffecb18d7dac ↩
5b51c1159445007956f095a4f2c.nt
```

The placeholders in the file name, such as `-P-` and `-O-`, are used to denote whether the hashed node is a subject, predicate or object. Standard HTTP query strings were not used in the file name to denote the type of a node, because most Web servers are not configured to obtain static resources using query strings.

2.3 Performance and Limitations

Storing all the generated files in a single directory does not perform well. LDstatic adopts a different strategy where each file name is split into an array of at least 10 characters. For each element of this array a directory is created forming a path, and the original file is stored under the generated path. This allows us to spread the files into an hierarchy of different directories rather than storing them within the same directory.

Our tests showed that the files were generated quickly using small data sources with around 1,000 triples. However, testing large data sets with more than 1,000,000 triples resulted in the creation of too many files that were impractical to handle by an operating system. An important limitation to keep in mind is the file repository disk quota. GitHub Pages currently has a 1GB limit per repository. As a consequence, given that there is a file for every possible request pattern, the size of an LDstatic repository is drastically larger than the

original RDF source: an RDF/XML source of 21 Kilobytes in size, was compiled into static resources reaching a total size of 4.1 Megabytes.

The performance and the availability of the published linked data were positive. `http://lmatteis.github.io/ldstatic/` is an online example of LDstatic which shows how the underlying TPFs can be queried using a TPF browser-based client called Client.js[6]. Fragment requests never took longer than 170 milliseconds.

2.4 How to use LDstatic

LDstatic requires Java 1.6.x or greater to run. All the other dependencies are contained within the downloaded package. Follow these steps to run LDstatic:

1. Download the source-code from: `https://github.com/lmatteis/ldstatic`
2. Run the bash script to generate the linked data. The first argument is the path to the RDF source file. The second argument is the namespace used for the fragments metadata and controls:
   `sh build.sh foaf.rdf http://lmatteis.github.io/ldstatic/ldf/`
3. The `ldf/` directory will now contain your static linked data resources that can be published on GitHub Pages or other file hosting repositories.

3 LDF-GAE

We will now describe the functionality of LDF-GAE[7], a tool we have developed to facilitate the hosting of linked data on Google's servers. Specifically, it is an implementation of a Triple Pattern Fragment server using the Google App Engine[8] cloud computing platform. App Engine is a platform-as-a-service hosting solution which abstracts all of the operating system details in favor of a set of language-specific APIs [9]. It supports various languages, as well as Java, therefore Apache Jena was used to parse RDF data sources.

3.1 Storing Triples on the Google App Engine Datastore

The main challenge of implementing a TPF server on App Engine was trying to find an efficient way of storing triples using the App Engine datastore. Unlike other traditional datastores, the App Engine datastore supports sets of properties in data objects also known as "entities" [9]. Data can be extracted from these objects using queries filtered by property values.

We represent a triple statement using the following entity structure:

[6] `https://github.com/LinkedDataFragments/Client.js`
[7] `https://github.com/lmatteis/ldf-gae`
[8] `https://developers.google.com/appengine/`


```
{
  "subject"  : triple.getSubject().toString(),
  "predicate": triple.getPredicate().toString(),
  "object"   : triple.getObject().toString()
}
```

The `triple` variable is an instance of the `com.hp.hpl.jena.graph.Triple` class. We call `.toString()` to obtain the literal representation of the node. A string value is then stored in datastore. We chose this entity structure so that we could directly match a triple pattern selector using App Engine's filtering capabilities. The nodes are serialized using Jena's lexical representation, and can therefore be unserialized into more complex Java objects, such as instances of `com.hp.hpl.jena.graph.Node`, using `NodeFactory.createLiteral(String)`.

3.2 Performance and Limitations

The RDF parsing routine is initialized by performing an `HTTP GET` request on `/parse`. A remote procedure call (RPC) to datastore is then executed for every triple found in the original data source. The distributed nature of App Engine enables the parsing routine to run using several backend instances concurrently, thus enabling parsing and storing data more efficiently.

To match a specific triple pattern, a remote procedure call is performed against the datastore to filter based on the properties. `http://ldf-gae.appspot.com/` is an online example of LDF-GAE running directly on App Engine's platform. Our tests showed that running a request to datastore to match a triple pattern takes on average 200 milliseconds.

App Engine imposes a certain number of limitations on the services it offers. Specifically for free users there currently is a 1GB limit on the total size of the data stored within datastore.

3.3 How to use LDF-GAE

LDF-GAE requires Java 1.7.x or greater, and the latest Java App Engine SDK[9] to run. All the other dependencies are contained within the downloaded package. Follow these steps to run LDF-GAE:

1. Download the source-code from: `https://github.com/lmatteis/ldf-gae`
2. Store the RDF sources you wish to parse in the `ldf-gae/rdf/` directory.
3. The `ldf-gae/` folder acts as a standard App Engine webapp. Follow the App Engine instructions to deploy it to the App Engine cloud: `https://developers.google.com/appengine/docs/java/`
4. Once the app is uploaded to App Engine, initiate the RDF parsing routine by accessing the `/parse` page: `http://<app-id>.appspot.com/parse`
5. Finally, access the root page to start querying the TPF server: `http://<app-id>.appspot.com/`

[9] `https://developers.google.com/appengine/downloads`

4 Conclusion

We have described the functionality of two separate tools, LDstatic and LDF-GAE, that enable linked data to be published for free and with high availability. LDstatic can be used to publish linked data on static Web servers, while LDF-GAE can be hosted on Google's computing platform. We started with examining various linked data partitioning strategies such as SPARQL endpoints and RDF dumps, but they either suffer from low availability or large granularity. To solve this, we used the Triple Pattern Fragments partitioning strategy to publish linked data, and proved that this type of linked data can be published with high availability on low-cost Web servers. These types of fragments are simple to serve and require little resources, giving them higher availability. Although free hosting services have various storage limitations, TPFs can still be distributed across several providers and intelligent agents can perform queries against them. TPF servers are therefore straightforward to implement on top of existing storage solutions, ranging from basic file repositories to highly scalable cloud platforms.

References

1. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data-The Story So Far. *International journal on semantic web and information systems*, 5(3):1–22, 2009. `http://eprints.soton.ac.uk/271286/1/paper00.pdf`.
2. Chris Bizer, Richard Cyganiak, Tom Heath, et al. How to Publish Linked Data on the Web. 2007. `http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/LinkedDataTutorial/`.
3. Olaf Hartig and Johann-Christoph Freytag. Foundations of Traversal Based Query Execution over Linked Data. In *Proceedings of the 23rd ACM conference on Hypertext and social media*, pages 43–52. ACM, 2012. `http://www.dbis.informatik.hu-berlin.de/fileadmin/research/papers/conferences/2012_hypertext_hartig_preprint.pdf`.
4. Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. *Executing SPARQL Queries over the Web of Linked Data*. Springer, 2009. `https://www2.informatik.hu-berlin.de/~hartig/files/HartigEtAl_QueryTheWeb_ISWC09_Preprint.pdf`.
5. Carlos Buil-Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. SPARQL Web-Querying Infrastructure: Ready for Action? In *The Semantic Web–ISWC 2013*, pages 277–293. Springer, 2013. `http://vmwebsrv01.deri.ie/sites/default/files/publications/paperiswc.pdf`.
6. Ruben Verborgh, Miel Vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik Van de Walle. Web-Scale Querying through Linked Data Fragments. In *Proceedings of the 7th Workshop on Linked Data on the Web*, 2014. `http://linkeddatafragments.org/publications/ldow2014.pdf`.
7. Ruben Verborgh. Triple Pattern Fragments. 2014. `http://www.hydra-cg.com/spec/latest/triple-pattern-fragments/`.
8. Wikipedia. Comparison of filename limitations. `http://en.wikipedia.org/wiki/Filename#Comparison_of_filename_limitations`, 2014. [Online; accessed 04-July-2014].
9. Alexander Zahariev. Google App Engine. *Helsinki University of Technology*, 2009. `http://www.cse.hut.fi/en/publications/B/5/papers/1Zahariev_final.pdf`.

