LDCache – a cache for Linked Data-driven Web applications

Hannes Ebner[1,2] and Matthias Palmér[1]

[1] MetaSolutions AB, Sweden
{hannes, matthias}@metasolutions.se
[2] KTH Royal Institute of Technology, Sweden

Abstract. Modern Web applications that make use of external data sources, be it Linked Data (LD) or not, usually run into the same problems; if a data source performs badly or is offline, the user experience is affected negatively. In some cases this may result in long response times, whereas in more extreme cases the application becomes unusable. This paper presents LDCache [4], a caching service that ensures that Linked Data-driven Web applications remain functional with good user experience regardless of the status of the external data sources they eventually integrate. First, some requirements are defined that typically occur when data from disparate sources are integrated. Then the features of the implemented solution are summarized together with documentation on how the service can be taken advantage of. This is followed by a description of the architecture. The authors conclude with a road map for future development and a short summary of the work carried out.

1 Background

When implementing Web applications that take advantage of multiple distributed data sources a range of problems have to be solved to maintain an acceptable user experience. Performance, scalability and reliability are important aspects that must not be ignored during the design and development process.

Because of the use of HTTP it is possible to cover a majority of the requirements (see following section for details) by taking full advantage of the mature HTTP tooling ecosystem; reverse proxies, HTTP caches, Content Delivery Networks (CDN), mirroring of datasets by importing RDF dumps into local instances, etc., are ways to gain control and increase reliability and performance. There has also been some work on solving slightly related client-side problems such as the *Semantic Web Client Library* [14]. The *Linked Data Fragments* initiative [5] is based on subsets of Linked Data collections which are queried on the client side. However, nothing really allows for the dynamic optimization of the usage of a diverse range of only minor parts of possibly large datasets. Datasets such as DBpedia [2] bring a large overhead at a high cost if they are to be self-hosted while most probably only using a subset of the available data.

LDCache aims to be a light-weight solution that implements a set of features that may be used by typical LD-driven applications. It assists in traversing

(parts of) the Linked Data graph by following links and provides a pre-caching and filtering mechanism to manage data that is relevant for particular client-side applications. Such an approach encourages to couple an LD-driven application with an LDCache configuration that exactly matches the application's requirements. The technical requirements of the necessary infrastructure can be kept at a minimum. To clarify LDCache's behavior in comparison with e.g. caching HTTP proxies it has to be emphasized that no requests to the source servers are performed during normal cache operation. Due to the currently applied pre-caching mechanism LDCache only returns data that has been cached in advance, which is something to take into consideration if frequently updated data is to be cached.

2 Requirements of typical LD-driven applications

The following requirements have been formalized during the development of several interactive Web applications that take advantage of Linked Data from third parties. The requirements are listed and described individually, but typical scenarios require combinations of one or more of them.

2.1 Reliability issues of third party data sources

It is common that third party data sources have regular down times or inconsistent performance. Especially big data hubs with a high load such as DBpedia are prone to being unreliable. Applications in which information from different sources is mashed up can be prepared in various ways for the unavailability of one or more data sources: (1) the application may degrade gracefully and still work, showing less information, (2) the application breaks and stops working, or (3) the application uses a local copy of (parts of) the data source and is not affected by any reliability issues. LDCache provides an easy to use solution for (3).

2.2 Streamlining multiple dereferenciation steps

Dereferenciation introduces complexity; redirects have to be followed which causes multiple requests that most probably cause delays that affect interactive applications negatively. With LDCache only a single request is necessary as the dereferenciation steps are streamlined on the server-side into one response.

2.3 RDF format alignment

If an application integrates different data sources then the support of various RDF formats may become an issue as more complex content negotiation scenarios apply and JavaScript-libraries have to be added to support additional formats. With LDCache as an intermediate layer a Web application can use the same content type for all RDF resources, even if the desired format is not directly supported by data source. E.g., an application can rely on *JSON-LD*, even if the data source only supports *RDF/XML*.

2.4 Offloading of Web resources to a CDN

It is common to make use of a CDN for the most requested content in high load scenarios. This requires that the DNS of the server can be controlled, which usually is not the case for third party data sources. LDCache makes it possible to provide URLs for cached information and allows for using a CDN where the URLs and their URL-parameters usually are changed into canonical representations.

3 Features and usage

3.1 Graph traversal

The configuration allows to define *databundles* consisting of lists of resources that are to be used as starting point for the caching process which is initiated during the startup of the service, i.e., the cache is pre-populated. Initially all resources are fetched from the provided URIs; if the graph traversal's *depth* parameter is greater than zero, a depth-first search (DFS) is performed (loops are detected). If *follow* and/or *followTuples* parameters are provided then resources in object or subject positions, respectively, are followed. This process has implications for the definition of an LDCache databundle; because links to other sources are followed, an LDCache databundle is not limited to a single data source and may contain resources from multiple LD datasets.

3.2 Cached resources

The REST resource that accesses the cache requires that the requested RDF resources (and subsequent resources if links are to be followed) are cached in the local LDCache repository. See the configuration section of the LDCache documentation [4] for instructions on how to configure LDCache to prefetch RDF resources. The request URL for cached resources is at the root of the LDCache: `http://{ldc-base}/`.

The following parameters are available when accessing cached resources:

- *url*: The RDF resource to be fetched.
- *follow*: A comma-separated list of predicates. The objects of matching triples are followed during graph traversal and if these objects correspond to cached resources their RDF graphs will be merged into the response graph.
- *followTuples*: A comma-separated list of *predicate-object tuples* in the format `predicate|object`, i.e., the predicate is separated from the object by the pipe symbol. The subjects of matching triples are followed.
- *followDepth*: The maximum distance from the root-resource that should be followed. Default is 0, i.e., only the resource identified by the url parameter will be fetched and no links are followed.
- *includeDestinations*: A comma-separated whitelist with prefixes of destinations to include when traversing the graph.

– *includeLiteralLanguages*: A comma-separated whitelist with language tags to specify which literals to include.
– *callback*: Name of the callback method to be used for a JSONP response.
– *format*: A valid RDF MIME type. If an *Accept* header is supplied the format parameter takes precedence.

The parameters *follow* and *followTuples* may be used in combination. Namespace expansion is applied to all parameters, refer to the LDCache documentation which contains a list of supported prefixes.

It has to be emphasized that the configuration parameters match the available request parameters of the API. In consequence the cache does not return satisfying results for requests that are out of scope of the cached data. However, the proxy resource may be used for such out-of-scope requests.

3.3 Uncached (proxied) resources

The proxy service bypasses the cache and allows arbitrary RDF resources to be fetched. Format conversions are handled transparently through normal content negotiation between proxy and data source and proxy and client. The proxy resource is quite simple as it only proxies one resource without the smartness that the cache provides. The proxy has to be explicitly activated in the configuration and is, like the cache, restricted to RDF resources.

The request URL of the proxy is: `http://{ldc-base}/proxy`.

3.4 Implicit dereferenciation

A common issue in LD-based applications are dereferenciations. In combination with following links to other resources and consecutive redirects this can cause unacceptable delays in user interfaces. LDCache flattens redirects automatically, e.g., when requesting `http://dbpedia.org/resource/Marie_Curie` in format N3 the response from LDCache corresponds to `http://dbpedia.org/data/Marie_Curie.n3`. No redirects are necessary and the client receives data that can be used immediately. This works for the whole traversal path in case any links to other resources are to be followed in the context of the request.

3.5 Rate limitation of HTTP requests

LDCache supports the limitation of requests per host to avoid overloading servers which may lead to being blacklisted. This feature is configurable.

3.6 Transparent handling of RDF formats

All content negotiation is handled transparently. During the caching process LDCache negotiates with the RDF source; when a client requests a cached resource from LDCache any RDF format supported by Sesame Rio [11] plus *JSON-LD* [3] is supported. This also means that LDCache may be used as transparent RDF converter in case an RDF source does not support the desired RDF format directly.

4 Example requests

The requests below illustrate how resources may be fetched, including related resources that are pointed out by outgoing links. The requests require that all data are prefetched and cached as there will be no on-demand requests to any data source. All matching resources are merged into a single RDF graph which then is returned to the requesting client.

4.1 Marie Curie, her doctoral students and their doctoral students

Outgoing links to resources in object position are followed, but restricted to doctoral students that have are identified by a URI that starts with dbpedia.org. The graph is traversed two levels down, the resource graphs are merged and the result is returned in JSON-LD.

```
GET http://{ldc-base}/?
    url=http://dbpedia.org/resource/Marie_Curie&
    follow=http://dbpedia.org/ontology/doctoralStudent&
    followDepth=2&
    includeDestinations=http://dbpedia.org/&
    format=application/ld+json
```

4.2 All Nobel laureates by following matching resources in subject position

All outgoing links are followed because the *follow* parameter (acting as whitelist) is omitted. In addition to following all resources in object position, the information from the *followTuples* parameter (which is a tuple consisting of predicate and object) is used to follow matching resources in subject position. The graph is traversed one level down, the graphs are merged and the result is returned in *Turtle* format.

```
GET http://{ldc-base}/?
    url=http://data.nobelprize.org/all/laureate&
    followTuples=rdf:type|nobel:Laureate&
    followDepth=1&
    format=text/turtle
```

4.3 Returning only partial data

In the next milestone support for a filter parameter will be added. This allows to reduce the size of responses by only returning a set of triples as defined by the filter parameter. This is useful for cases where e.g. only labels are relevant for the requesting application. In such situations it will be possible to only request e.g. *rdfs:label*, *skos:prefLabel*, *dc:title*, and *dcterms:title*; all other triples will be omitted.

5 Architecture

The architecture is kept simple and consists of few components:

- A *Restlet*-supported API [12] is wrapped around a *Sesame SAIL* [11].
- The application can be started either stand-alone (supported by the Simple framework [13]) or as web app in a container using Tomcat, Jetty, etc.
- Every cached RDF resource is stored as one *named graph* [1].
- Administrative information about datasets is stored in separate named graphs.
- The API is intentionally kept simple and currently contains only two REST resources: one for the cache and one for the proxy.
- The configuration format is JSON.

6 Road map for future development

The development road map contains some major features and improvements:

- Continuous background updates of databundles and their cached resources.
- Enhanced property filtering to be smart about storing "interesting" values
- On-the-fly caching of proxied resources; proxied RDF resources are currently discarded after a proxy request.
- Support for SPARQL queries to find a list of resources that should be cached.
- Explore whether support for Linked Data Fragments [5] would make sense.
- Investigate how SPARQL requests could be cached.

7 Conclusions

LDCache has been designed as a light-weight and easy to use Linked Data cache that supports the development of Linked Data-driven Web applications. The current version is still in its early stages with room for improvements, however, it is considered stable and safe to use in production environments. A showcase where LDCache is used in production are Nobel Media's [8] profile pages of Nobel laureates that consist of information from various data sources, e.g., Nobel Prize Linked Data [6,7] and DBpedia [2].

The goal of fulfilling the stated requirements has been met and the authors feel that LDCache improves the lead time for developing Linked Data-driven applications. Also, it improves the performance, scalability, and reliability of applications that rely on third party data sources and brings back control of the data to the application developer.

Acknowledgements

The work presented in this paper has been partially carried out with financial support from the VINNOVA-funded project "Vidareutveckling av länkade öppna data för Nobelpris" [9] and the EC-funded project "Open Discovery Space" [10] which the authors gratefully acknowledge.

References

1. Carroll, J., Bizer, C., Hayes, P., Stickler, P.: Named graphs. Web Semantics: Science, Services and Agents on the World Wide Web 3(4), 247–267 (2005)
2. DBpedia (2014), http://dbpedia.org
3. JSON-LD – JSON for Linking Data (2014), http://json-ld.org, accessed 2014-07-25
4. LDCache documentation (2014), http://entrystore.org/ldcache/
5. Linked Data Fragments (2014), http://linkeddatafragments.org/, accessed 2014-07-24
6. Nobel Prize Linked Open Data (2014), http://data.nobelprize.org, accessed 2014-07-24
7. Specification of Nobel Prize Linked Data (2013), http://www.nobelprize.org/nobel_organizations/nobelmedia/nobelprize_org/developer/manual-linkeddata/terms.html, accessed 2014-07-24
8. Nobel Media (2014), http://www.nobelprize.org/nobel_organizations/nobelmedia/, accessed 2014-07-24
9. Vidareutveckling av länkade öppna data för Nobelpris (2014), http://www.vinnova.se/sv/Resultat/Projekt/Effekta/2012-00741/Vidareutveckling-av-lankade-oppna-data-for-Nobelpris/, accessed 2014-07-24
10. Open Discovery Space (2014), http://opendiscoveryspace.eu
11. OpenRDF Sesame (2014), http://www.openrdf.org/, accessed 2014-07-24
12. Restlet – REST framework for Java (2014), http://restlet.com/, accessed 2014-07-24
13. Simple Framework (2014), http://www.simpleframework.org/, accessed 2014-07-24
14. Semantic Web Client Library – Querying the complete Semantic Web with SPARQL (2009), http://wifo5-03.informatik.uni-mannheim.de/bizer/ng4j/semwebclient/, accessed 2014-07-24

