LDSpider: An open-source crawling framework for the Web of Linked Data

Robert Isele[3], Jürgen Umbrich[2], Christian Bizer[3], and Andreas Harth[1]

[1] AIFB, Karlsruhe Institute of Technology
harth@kit.edu

[2] Digital Enterprise Research Institute, National University of Ireland, Galway
juergen.umbrich@deri.org

[3] Freie Universität Berlin, Web-based Systems Group
robertisele@googlemail.com, chris@bizer.de

Abstract. The Web of Linked Data is growing and currently consists of several hundred interconnected data sources altogether serving over 25 billion RDF triples to the Web. What has hampered the exploitation of this global dataspace up till now is the lack of an open-source Linked Data crawler which can be employed by Linked Data applications to localize (parts of) the dataspace for further processing. With LDSpider, we are closing this gap in the landscape of publicly available Linked Data tools. LDSpider traverses the Web of Linked Data by following RDF links between data items, it supports different crawling strategies and allows crawled data to be stored either in files or in an RDF store.

Keywords: Linked Data, Crawler, Spider, Linked Data tools

1 Introduction

As of September 2010, the Web of Linked Data contains more than 200 interconnected data sources totaling in over 25 billion RDF triples[4]. Applications that need to localize data from the Web of Linked Data [1] for further processing currently either need to implement their own crawling code or rely on pre-crawled data provided by Linked Data search engines or in the form of data dumps, for example the Billion Triples Challenge dataset[5]. With LDSpider, we are closing this gap in the Linked Data tool landscape. LDSpider is an extensible Linked Data crawling framework, enabling client applications to traverse and to consume the Web of Linked Data.

The main features of LDSpider are:

- LDSpider can process a variety of Web data formats including RDF/XML, Turtle, Notation 3, RDFa and many microformats by providing a plugin architecture to support Any23[6].

[4] http://lod-cloud.net
[5] http://challenge.semanticweb.org/
[6] http://any23.org/

- Crawled data can be stored together with provenance meta-information either in a file or via SPARQL/Update in an RDF store.
- LDSpider offers different crawling strategies, such as breadth-first traversal and load-balancing, for following RDF links between data items.
- Besides of being usable as a command line application, LDSpider also offers a simple API which allows applications to configure and control the details of the crawling process.
- The framework is delivered as a small and compact jar with a minimum of external dependencies.
- The crawler is high-performing by employing a multi-threaded architecture.

LDSpider can be downloaded from Google Code[7] under the terms of the GNU General Public License v3. In the following, we will give an overview of the LDSpider crawling framework and report about several use cases in which we employed the framework.

2 Using LDSpider

LDSpider has been developed to provide a flexible Linked Data crawling framework, which can be customized and extended by client applications. The framework is implemented in Java and can be used through a command line application as well as a flexible API.

2.1 Using the command line application

The crawling process starts with a set of seed URIs. The order how LDSpider traverses the graph starting from these seed URIs is specified by the crawling strategy. LDSpider provides two different round-based crawling strategies:

The breadth-first strategy takes three parameters: `<depth> <uri-limit> <pld-limit>`. In each round, LDSpider fetches all URIs extracted from the content of the URIs of the previous round, before advancing to the next round. The depth of the breadth-first traversal, the maximum number of URIs crawled per round and per pay-level[8] domain as well as the maximum number of crawled pay-level domains can be specified. This strategy can be used in situations where only a limited graph around the seed URIs should be retrieved.

The load-balancing strategy takes a single parameter: `<max-uris>`. This strategy tries to fetch the specified number of URIs as quickly as possible while adhering to a minimum and maximum delay between two successive requests to the same pay-level domain. The load-balancing strategy is useful in situations where the fetched documents should be distributed between domains without overloading a specific server.

[7] `http://code.google.com/p/ldspider/`

[8] "A pay-level domain (PLD) is any domain that requires payment at a TLD or cc-TLD registrar."[3]

LDSpider will fetch URIs in parallel employing multiple threads. The strategy can be requested to stay on the domains of the seed URIs.

Crawled data can be written to different sinks: File output writes the crawled statements to files using the N-Quads format. Triple store output writes the crawled statements to endpoints that support SPARQL/Update.

2.2 Using the API

LDSpider offers a flexible API to be used in client applications. Each component in the fetching pipeline can be configured by either using one of the implementations already included in LDSpider or by providing a custom implementation. The fetching pipeline consists of the following components:

The Fetch Filter determines whether a particular page should be fetched by the crawler. Typically, this is used to restrict the MIME types of the pages which are crawled (e.g. to RDF/XML).

The Content Handler receives the document and tries to extract RDF data from it. LDSpider includes a content handler for documents formatted in RDF/XML and a general content handler, which forwards the documents to an Any23 server to handle other types of documents including Turtle, Notation 3, RDFa and many microformats.

The Sink receives the extracted statements from the content handler and processes them usually by writing them to some output. LDSpider includes sinks for writing various formats including N-Quads and RDF/XML as well as to write directly to a triple store using SPARQL/Update. Both sinks can be configured to write metadata containing the provenance of the extracted statements. When writing to a triple store, the sink can be configured to include the provenance using a Named Graph layout.

The Link Filter receives the parsed statements from the content handler and extracts all links which should be fetched in the next round. A common use of a link filter is to restrict crawling to a specific domain. Each Link Filter can be configured to follow only ABox and/or TBox links. This can be used for example to configure the crawler to get the schema together with the primary data.

2.3 Implementation

LDSpider is implemented in Java and uses 3 external libraries: The parsing of RDF/XML, N-Triples and N-Quads is provided by the NxParser library[9]. The HTTP functionality is provided by the Apache HttpClient Library[10], while the Robot Exclusion Standard is repected through the use of the Norbert[11] library.

[9] `http://sw.deri.org/2006/08/nxparser/`
[10] `http://hc.apache.org/`
[11] `http://www.osjava.org/norbert/`

3 Usage examples

We have employed LDSpider for the following crawling tasks:

- We employed LDSpider to crawl interlinked FOAF profiles and write them to a triple store. For that purpose, we crawled the graph around a single seed profile (`http://www.wiwiss.fu-berlin.de/suhl/bizer/foaf.rdf`) and compared the number of traversed FOAF profiles for different number of rounds:

rounds	1	2	3	4	5
profiles	1	10	101	507	6730

- We employed LDSpider to crawl Twitter profiles, which expose structured data using RDFa. We started with a single seed profile (`http://twitter.com/aharth`) and wrote all traversed profiles to a triple store and compared the number of profiles for different number of rounds:

rounds	1	2	3
profiles	1	38	1160

  As the number of profiles grows faster than in the previsous use case, we can conclude that the interlinked Twitter profiles build a much denser graph than the FOAF web.
- LDSpider is used in an online service which executes live SPARQL queries over the LOD Web[12]
- We used LDSpider to gather datasets for various research projects; e.g. the study of link dynamics [4] or the evaluation of SPARQL queries with data summaries over Web data [2]

In summary, LDSpider can be used to collect small to medium-sized Linked Data corpora up to hundreds of millions of triples.

References

1. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.
2. Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. Data summaries for on-demand queries over linked data. In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 411–420, New York, NY, USA, 2010. ACM.
3. Hsin-Tsang Lee, Derek Leonard, Xiaoming Wang, and Dmitri Loguinov. Irlbot: scaling to 6 billion pages and beyond. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 427–436, New York, NY, USA, 2008. ACM.
4. Michael; Hogan Aidan; Polleres Axel; Decker Stefan Umbrich, Jürgen; Hausenblas. Towards dataset dynamics: Change frequency of linked open data sources. *3rd International Workshop on Linked Data on the Web (LDOW2010), in conjunction with 19th International World Wide Web Conference*, 2010.

[12] `http://swse.deri.org/lodq`