Computing Linked Data On-demand Using the VOLT Proxy

Blake Regalia and Krzysztof Janowicz

STKO Lab, University of California, Santa Barbara, USA
blake@geog.ucsb.edu, janowicz@ucsb.edu

Abstract. The Linked Data paradigm has changed how data on the Web is published, retrieved, and interlinked, thereby enabling modern question answering systems and contributing to the spread of open data. With the increasing size, interlinkage, and complexity of the Linked Data cloud, the focus is now shifting towards strategies and technologies to ensure that Linked Data can also succeed as an infrastructure. This raises questions about the sustainability of query endpoints, the reproducibility of scientific experiments conducted using Linked Data, the lack of established quality metrics, as well as the need for improved ontology alignment and query federation techniques. One core issue that needs to be addressed is the trade-off between storing data and computing them on-demand. Data that is derived from already stored data, changes frequently in space and time, or is the output of some workflow, should be computed. However, such functionality is not readily available on the Linked Data cloud today. To address this issue, we have developed a transparent SPARQL proxy that enables the on-demand computation of Linked Data together with the provenance information required to understand how the data were derived. Here, we demonstrate how the proxy works under the hood by applying it to the computation of cardinal directions between geographic features in DBpedia.

1 Motivation

Despite the rapid growth of Linked Data and tool chains for publishing, storing, interlinking, and querying Linked Data, large-scale, real-world success stories are emerging slowly. There are many reasons for this, e.g., the difficulty of maintaining a scale-able SPARQL endpoint, and several approaches have been introduced within the last few years to make the Linked Data cloud more sustainable [2].

We believe that one challenge that has not been addressed sufficiently is the accuracy, completeness, and up-to-dateness of the data as such. From a user's perspective, it is often unclear why certain data are present while other data are not, how much uncertainty is inherent in the queried (RDF) statements, and how stable these statements are. More technically speaking, parts of this challenge are rooted in the open question of which Linked Data triples should be stored and which should be computed. Intuitively, data that is derived from already stored data, that changes frequently in space and time, or is the result of a sequence of workflow steps, should be computed. To give a simple example,

a property such as population density can be computed based on the area and population. In contrast, if the density is stored, it needs to be kept in sync with the changing population (and area) especially if multiple population values are given, e.g., from different years or on the municipality and the metropolitan level.

A similar example can be given for the completeness and accuracy cases. DBpedia stores 133,941 cardinal direction triples such as `dbr:San_Francisco dbp:northeast dbr:Berkeley,_California`. Why those triples exist and all the billions of cardinal direction triples between other pairs of places do not, remains unclear to the end user. Clearly, due to combinatorial explosion, it is impossible to store all possible triples so they should instead be computed on-demand. Furthermore, a simple experiment reveals that more than half of all cardinal direction relations currently stored in DBpedia are invalid or incorrect [1].

To address these and related challenges, we have developed a transparent proxy that can sit in front of arbitrary SPARQL 1.1 endpoints to augment them with computational capabilities by deriving triples and provenance records on-demand.

In this demonstration, we show how to: define a *computable property* procedure using the VOLT language, compile it to RDF, invoke computation on the procedure using real data from DBpedia, and inspect the results along with their provenance metadata. The full video is available at: `https://youtu.be/EO2cD6Qy-Hc`.

2 Demonstrating VOLT

In this section, we demonstrate how the transparent VOLT proxy works and how it interacts with arbitrary SPARQL endpoints by applying it to the cardinal directions use case introduced before.

2.1 Interface and Encapsulation

The VOLT proxy behaves transparently, making it appear to the user as though they are directly querying the encapsulated SPARQL endpoint. However, a single query issued by the end-user may prompt several interactions between the proxy and the actual endpoint. As depicted in Figure 1, results from the original input query are directly returned to the client.

2.2 The VOLT Language and Compiler

The VOLT proxy requires RDF statements in the triplestore's *model graph* to follow a strict ontology because those triples ultimately get translated into machine code. Since the number of triples and nested blank nodes grows quickly with the more sophisticated procedures, directly encoding procedural logic into RDF can be a tedious and error-prone task for humans. Similar to the reasons that programming languages were invented to provide a layer of abstraction between the developer and assembly code, we have created a custom abstraction language designed specifically to streamline the process of coding procedures and their

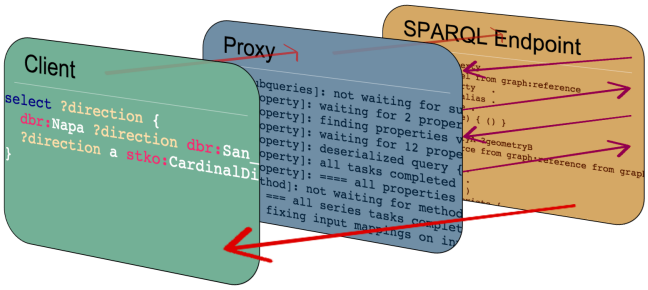

Fig. 1 A depiction of the interaction that occurs between the client, VOLT proxy and encapsulated SPARQL endpoint.

embedded SPARQL queries. Aside from significantly reducing the amount of code a developer has to write as well as improving its readability, the compiler also performs optimizations such as data locality, parallelization, refactoring, strength reduction, and so forth.

2.3 Creating a Procedure

For this demonstration, we start by defining a *computable property* using the VOLT programming language. A computable property represents a potential relation between two individuals. It defines all the criteria required by each the subject and object in order for a particular directed relation to hold between them. An *input triple* refers to a triple that functions as an input to a computable property. It has an explicit RDF term for each the subject, predicate, and object (i.e., it is not a *pattern*). We start the computable property definition by declaring its name as an IRI, which acts as the predicate of an input triple anytime the procedure gets invoked by a SPARQL query. In this demonstration, the computable property is named `stko:south`, which will test if the object of an input triple is south of its subject.

The PostGIS[1] plugin we use for this demonstration simply translates Extensible Value Testing (EVT) function calls on RDF literals from the SPARQL query into SQL strings and pipes them into a PostgreSQL[2] child process on the host machine, enabling a user to access the full domain of spatial functions provided by PostGIS. For computing cardinal directions, we employ the ST_Azimuth[3] function to compute the clockwise angle from A to B in radians relative to true north. With the help of some expressions, we can test if the radians returned by

[1] http://postgis.net/
[2] http://www.postgresql.org/
[3] http://postgis.net/docs/ST_Azimuth.html

the `postgis:azimuth` call falls within the range that our model will consider to be `stko:south` [1].

2.4 Embedded SPARQL Queries

In order to compute the azimuth between two places, we need to obtain their coordinates. Using DBpedia data, we can extract the Well-known Text (WKT) from the `geo:geometry` property contained by each of the places we want to compare. Instructing our procedure to fetch this data from the triplestore means we will need to execute a SPARQL query at runtime. The VOLT language has a few syntactic variations when it comes to embedding SPARQL queries into a procedure's code. In the video demonstration, we show the implicit syntax which is the simplest way to obtain a value from one of the subject's or object's own triples. In this case, we select the triple(s) having `geo:geometry` as the predicate. Since a subject or object may contain more than one triple that share the same predicate, VOLT will fork the execution of a procedure anytime a SPARQL query returns more than one result. This ensures that every combination of selections is treated to a procedure's computation. Also, if any subject or object in a computable property does not yield a result from the SPARQL query, for example a place does not have the `geo:geometry` property, execution on that input triple is cancelled.

2.5 Testing Relations Existentially

Computable properties can be used to explicitly test if a certain relation holds between two named individuals. For our example, we will be asking if San Diego is south of Yosemite National Park. The computable property we wrote, `stko:south`, will be evaluated using these two places as the object and subject, respectively. Since VOLT intends to act as transparently as possible, our modus operandi is to issue SPARQL queries as though we are querying for triples that may or may not exist in the triplestore. Therefore, to invoke computation on the computable property method we created, a client simply issues an ASK query containing a single triple whose predicate is the name of the property we want to test, `stko:south`, and puts the URIs for the places of Yosemite National Park and San Diego as the subject and object of the input triple. The benefit of this approach is that a user does not need to be aware that they may be invoking procedural computation in their query — although, they are able to discover all procedures that the proxy can offer. If a user intends to discover which relations exist between two entities, the proxy would have to test all possible procedures against that subject-object pair to determine which ones should be materialized. Such a request could cause unwanted delays during query execution. To address this issue, the proxy will only test procedures when an explicit pattern is stated in the query. This pattern requires a triple having a variable in the predicate position, where that variable also appears in a triple that asserts its `rdf:type`. The object of the predicate variable's `rdf:type` represents the class of procedures to invoke testing on. For example, `dbr:San_Diego ?rel dbr:Napa. ?rel a stko:CardinalDirection` tells the proxy to test all procedures that are declared

to be a type of `stko:CardinalDirection` and materialize the ones that evaluate positive.

If the input triple already exists in either the source graph (e.g., the relation is explicit) or the output graph (e.g., the relation was computed earlier), then the proxy does not evaluate the associated procedure since the solution is already present in the triplestore. On the other hand, if the computable property is determined to be viable for a given subject and object, i.e., if its procedure evaluates to `true`, then the input triple is materialized and stored to a temporary graph in the triplestore. By the time the proxy has finished evaluating all procedures invoked by the client's query, the proxy terminates its session by issuing the original SPARQL query on the union of the source graph and output graph(s), combining all source data with any triples that were derived by computation.

3 Summary and Outlook

In this demo paper he have showcased the transparent VOLT proxy by applying it to the computation of cardinal directions between geographic features in DBpedia. We have argued why it is important and often necessary to compute data on-demand instead of storing it and have highlighted some of the implementation details underlying VOLT. Future work will focus on improving the performance of the VOLT proxy (see [1] for first results), a library of commonly used computable properties for the geo-sciences and beyond, and on an ontological framework for caching.

References

1. Regalia, B., Janowicz, K., Gao, S.: VOLT: A Provenance-Producing, Transparent SPARQL Proxy for the On-Demand Computation of Linked Data and its Application to Spatiotemporally Dependent Data. In: M. D'Aquin, E. Blomqvist (eds.) ESWC 2016. Springer LNCS (2016, forthcoming)
2. Rietveld, L., Verborgh, R., Beek, W., Vander Sande, M., Schlobach, S.: Linked Data-as-a-Service: The Semantic Web Redeployed. In: The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia,, pp. 471–487. Springer (2015)