Decentralizing the Persistence and Querying of RDF Datasets Through Browser-Based Technologies

Blake Regalia - blake@geog.ucsb.edu

STKO Lab, University of California, Santa Barbara, USA

Abstract. Even though linked open data are inherently capable of persisting in a decentralized fashion, the Semantic Web community is currently lacking effective workflows for seamlessly sharing and querying RDF datasets beyond relatively small scales. The most accessible methods for cloning and subsequently querying an RDF dataset are quite involved, often requiring users to locate and download a data dump, install and run a local SPARQL engine, load data into the triplestore, and then query their endpoint using some interface. While these actions may qualify as the first step towards decentralization, the incentive for such a user to then make their local endpoint available to the World Wide Web and thus act as an auxiliary host or 'mirror' is hard to surmise. In this work, we answer the intelligent client challenge by presenting a browser-based RDF query engine coupled with web-based peer-to-peer networks to illustrate the decentralized persistence and use of RDF datasets via intelligent peers.

1 Introduction and Motivation

It is no surprise that in recent years, web services and standalone web applications have been introduced as viable competitors to desktop native applications thanks to innovative Web technologies [7]. When it comes to hosting and querying RDF datasets however, native applications are almost always involved, typically on the server side. While this technique is trivial for centralized architectures, it does not promote the decentralization of persisting and querying RDF datasets. [10] demonstrates the value that decentralization has on the Semantic Web by thoroughly examining the methods and applications found at the intersection of the Semantic Web and Peer-to-Peer. Additionally, as several previous works have demonstrated [12,13], *intelligent clients* play an important role when it comes to improving both the acessibility to RDF datasets as well as the reliability of obtaining query results. In this work, we strive towards the decentralization of persisting and querying RDF datasets by creating *intelligent peers* that may simultaneously act as data publishers, data hosts and data consumers.

2 Peer-to-Peer Communication

Using peer-to-peer (p2p) networks to disseminate large RDF datasets has two main advantages for the Semantic Web community: (1) data publishers who operate with limited resources may circumvent hosting costs once their dataset reaches a sufficient demand, i.e., a substantial number of *seeds*[1] and (2) data consumers should see an in-

[1] A peer that is uploading content for other peers

creased availability of datasets correlate with larger *swarm*[2] sizes. From the data publisher's perspective, a p2p content delivery network (CDN) operates at a fraction of the cost compared to the equivalent centralized approach because a p2p CDN only requires the cost of initial seeds. From the data consumer's perspective, as the number of seeds increases, so does the probability of faster content delivery for any new peer. While other works in the literature have focused on distributed RDF repositories [2,3,1,4], our approach is to deliver the entire dataset, or a complete selection of the dataset, to each peer so that applications can continue to function offline. We have also yet to see a p2p implementation of RDF persistence or querying in the browser.

Web Real-Time Communication (WebRTC)[3] is a set of protocols and APIs that is currently being standardized by W3C to enable communication between clients, within the browser, over p2p connections. A demonstration of this technology, already implemented in most major browsers, is made available by the open-source JavaScript library WebTorrent[4], which offers many of the same features as traditional torrent programs. For example, incoming torrent data are available to use on-arrival, meaning that applications can start operating on fragments of a dataset before the full file is downloaded. Although our prototype does not yet take advantage of this capability, a use case is demonstrated by the streaming video playback on the WebTorrent home page. We employ the WebTorrent library to demonstrate how WebRTC can be used to deliver RDF dataset content to clients within the browser.

3 Querying RDF Datasets in the Browser

While there are several existing browser-based triplestores and query engines [8,11,6,13], the major shortcomings of these JavaScript libraries have to do with the size of datasets they can reliably handle, query performance, or the overhead time of acquiring a dataset prior to querying. The cause for these shortcomings can be narrowed down to the fact that no client-side implementation is making use of a binary serialization format that applies *succinct data structures*. Such serializations are intended to create compact storage footprints while also enabling high-performance query operations without the need for decompression. HDT [5,9] is the pioneer binary serialization format for encoding RDF datasets using compact data structures that can be queried *in-place*, i.e., without transformations. In order to drive forward real-time Semantic Web applications that take place in the browser, intelligent clients should be able to quickly and efficiently transmit, receive and query RDF datasets with little overhead.

As a proof of concept, we provide here[5] a working prototype that combines our browser-based RDF query engine, a connection to a p2p network, and the use of RDF datasets that have been encoded using a custom binary serialization format based on HDT. The web application initially downloads an RDF dataset by connecting to a p2p network via WebTorrent. We don't go into details about our custom binary serialization format here, but the encoding software is written entirely in JavaScript and is therefore

[2] All active peers in a p2p network sharing the same torrent

[3] https://webrtc.org

[4] https://webtorrent.io

[5] http://phuzzy.link/query/

also capable of being used by intelligent peers to create new RDF datasets from within the browser. Result times will vary depending on the machine used to trial our web application demo, but we believe that its performance illustrates the potential that this type of approach has.

In its current state, our query engine works exclusively on basic graph patterns, i.e., sets of triple patterns and filters. In order to generate graph patterns, we designed a JavaScript API that follows a graph traversal query paradigm, much like the Gremlin Graph Traversal Language[6]. This low-level approach of using an API to build queries offers several compelling advantages to us over the more user-friendly alternative of a query language such as SPARQL. First of all, the data structures we are using to store RDF terms and RDF triples, as well as their complementary search algorithms, make the API-built queries straightforward to construct and evaluate. More importantly however, supporting a high-level query language such as SPARQL requires that some library will handle parsing, query planning and query optimization. With a graph traversal query paradigm, the costs associated with these processes are either voided or significantly mitigated by virtue of the query paradigm itself, e.g., parsing is handled by the runtime system directly and query planning is essentially performed by the query author. Reducing costs at this stage of query execution is pivotal to our goal of competing with the time it takes distant endpoints to return query results. In the broader scheme of providing a usable query engine, we wish to start by offering the bare essentials for querying in the form of an API. Support for a more advanced query language such as SPARQL is therefore supplemental and appropriate for future work. A sample query is shown in Listing 1, and its equivalent SPARQL query shown in Listing 2. Documentation about our query API is available online[7].

```
(graph) => graph.pattern()
   .object('dbo:City').edgeIn('rdf:type')
   .subjects().mark('city')
   .forkOut({
      'dbo:isPartOf': e => e.object('dbr:California'),
      'rdfs:label': e => e.literals()
         .data('language', k => k.in('en')).mark('name'),
      'dbo:populationTotal': e => e.literals()
         .data('number', k => k.is('>=',1e5).is('<',2e5).save('population')),
   })
   .exit();
```

Listing 1 A sample query that shows how our low-level JavaScript API is used to create graph patterns. This query selects all cities in California that have a population between 100,000 and 200,000 from a selection of the DBpedia dataset.

```
select * {
    ?city a dbo:City ;
        dbo:isPartOf dbr:California ;
        rdfs:label ?name ;
        dbo:populationTotal ?population .

    filter(isLiteral(?name))
    filter(langMatches(lang(?name), "en"))
    filter(isNumeric(?population))
    filter(?population >= 100000 && ?population < 200000)
}
```

[6] http://tinkerpop.apache.org/gremlin.html
[7] http://phuzzy.link/query/documentation

Listing 2 The equivalent SPARQL query to the graph pattern shown in Listing 1.

To illustrate how such an RDF query engine compares to a traditional triplestore, the query shown in Listing 1 yielded 45 results from a dataset consisting of 4.15 million triples (all *things* of type `dbo:City`) in $329ms$ on a modern laptop while consuming $421MiB$ in memory, shown in Figure 1, compared to the 45 results obtained in $365ms$ on a cold, local Fuseki instance with the exact same dataset while consuming $2.43GiB$ in memory. The most likely explanation for why we see comparable performance and a smaller memory footprint out of our browser-based query engine, and the reason that such comparisons are not fair to draw conclusions from, could be due to the fact that our system operates on a read-only snapshot of an efficiently packed dataset file with pre-built indexes. Nonetheless, the comparable query result times are a good indication that browser-based query engines may indeed have the strength to compete with distant endpoints where clients would normally be confronted by network latency, network throughput, and server-side resource limitations.

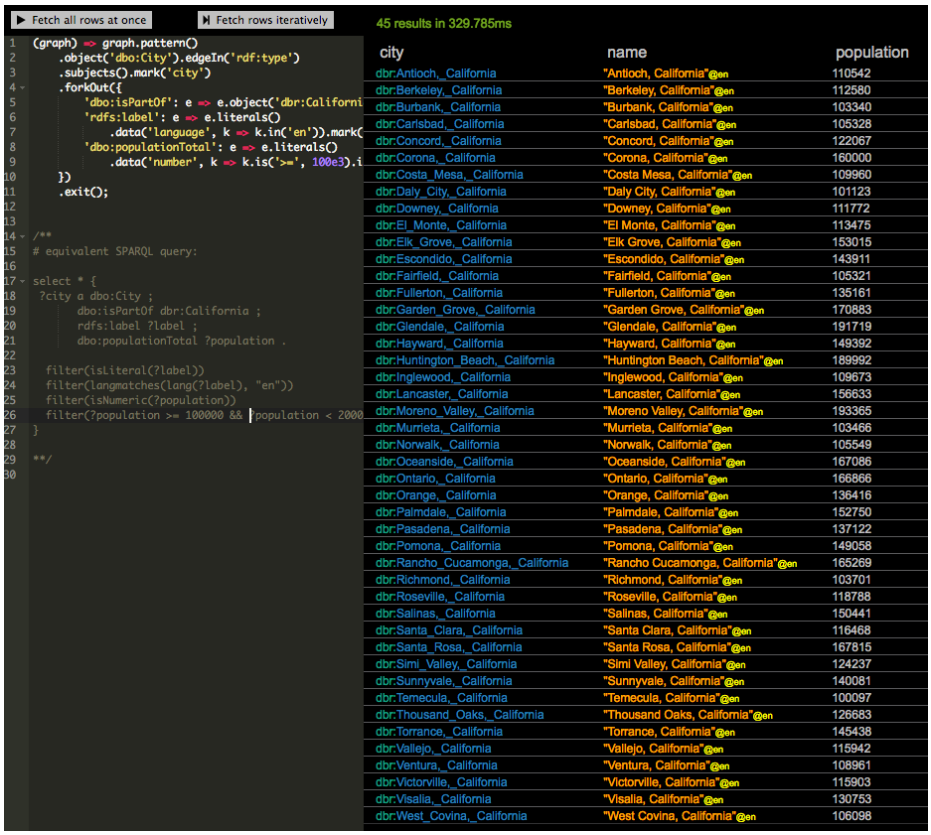

Fig. 1 A snapshot of the demo interface showing the sample query from Listing 1 on the left evaluated on a selection of the DBpedia dataset, consisting of 4.15 million triples, and the results shown on the right.

As these tools are still under active research and development, we are not yet ready to publish an evaluation of this work. However, we hope to demonstrate the potential implications that the associated methods have. The source code is made available under the development branch on GitHub[8].

4 Closing Remarks

In this work, we took steps towards decentralizing the Semantic Web from entirely within the browser. We combined the concepts of intelligent clients with peer-to-peer communication to create *intelligent peers*. We introduced our RDF query engine that operates on binary serializations of RDF in an effort to compete with the round trip times it takes distant endpoints to process and return query results to the client. We hope to continue our research in this direction so that datasets persisting in a decentralized architecture can come to fruition within the browser and so that they may be used and queried by data consumers offline.

References

1. Arumugam, M., Sheth, A.P., Arpinar, I.B.: Towards peer-to-peer semantic web: A distributed environment for sharing semantic knowledge on the web (2002)
2. Cai, M., Frank, M.: Rdfpeers: a scalable distributed rdf repository based on a structured peer-to-peer network. In: Proceedings of the 13th international conference on World Wide Web, pp. 650–657. ACM (2004)
3. Cai, M., Frank, M., Yan, B., MacGregor, R.: A subscribable peer-to-peer rdf repository for distributed metadata management. Web Semantics: Science, Services and Agents on the World Wide Web **2**(2), 109–130 (2004)
4. Dimartino, M.M., Calì, A., Poulovassilis, A., Wood, P.T.: Peer-to-peer semantic integration of linked data. In: CEUR Workshop Proceedings, vol. 1330, pp. 213–220. CEUR Workshop Proceedings (2015)
5. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary rdf representation for publication and exchange (hdt). Web Semantics: Science, Services and Agents on the World Wide Web **19**, 22–41 (2013)
6. Hernández, A.G., GARC1A, M.: A javascript rdf store and application library for linked data client applications. In: Devtracks of the, WWW2012, conference. Lyon, France (2012)
7. Kun, Y., Xiao-Ling, W., Ao-Ying, Z.: Underlying techniques for web services: A survey. In: Journal of software. Citeseer (2004)
8. Maccioni, A., Collina, M.: Graph databases in the browser: using levelgraph to explore new delhi. Proceedings of the VLDB Endowment **9**(13), 1469–1472 (2016)
9. Martínez-Prieto, M.A., Gallego, M.A., Fernández, J.D.: Exchange and consumption of huge rdf data. In: Extended Semantic Web Conference, pp. 437–452. Springer (2012)
10. Staab, S., Stuckenschmidt, H.: Semantic Web and Peer-to-peer: decentralized management and exchange of knowledge and information. Springer (2006)
11. Uchida, H., Swick, R., Sambra, A.: The web browser personalization with the client side triplestore. In: International Semantic Web Conference, pp. 470–485. Springer (2014)
12. Verborgh, R., Vander Sande, M., Colpaert, P., Coppens, S., Mannens, E., Van de Walle, R.: Web-scale querying through linked data fragments. In: LDOW (2014)

[8] https://github.com/blake-regalia/graphy.js/tree/development

13. Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., Colpaert, P.: Triple pattern fragments: A low-cost knowledge graph interface for the web. Web Semantics: Science, Services and Agents on the World Wide Web **37**, 184–206 (2016)