RDF Triple Stores for Client-side Applications

Joseph Utecht and Mathias Brochhausen PhD

Department of Biomedical Informatics, University of Arkansas for Medical Sciences, Little Rock, AR

1 Introduction

Client driven web applications are becoming increasingly popular [8]. At the same time REpresentation State Transfer (REST) protocals for datasets have become more standardized, allowing for Javascript frameworks to implement standardized create, read, update and delete (CRUD) functions. This presents some interesting problems when working with current RDF triple stores [7].

2 Material and Methods

2.1 RDF Triple Stores

We have decided to focus on Apache Jena [1], Blazegraph [2], and Sesame [4] RDF stores for our testing due to their support for RDFS reasoning, open source licenses, ability to handle large datasets and REST endpoints for interaction. [10]

Testing was performed on VirtualBox VM [6] with 2x 2.4 GHz 6-core Xeon processors and 8GB of ram. The operating system was CentOS 7 [3] and all of the stores were run through the Java application server Tomcat [5].

Apache Jena uses a Java based front-end called Fuseki these are distributed under the Apache License 2.0. Fuseki can either be run standalone or under a Java application server. For the testing the most recent version Fuseki 2.0.0 under Tomcat was used with RDFS level reasoning.

Systrap's Blazegraph (formerly known as Bigdata) is distributed under either the GPLv2 or a commercial license. It can also either be run as a standalone or in a Java application server. Blazegraph 1.5.1 under Tomcat was used for testing with a triple + inference graph.

Much like Jena, Sesame is a framework that also includes a web front-end and native triple store. Sesame version 2.8.4 was used for testing with a native triple store with RDFS inference.

2.2 Testing Method

We used a modified version of the Lehigh University Benchmark (LUBM) [9] to test performance and capability of the triple stores. We did not use the default testing queries with this dataset as they were more designed to measure the performance of OWL inference models, however this benchmark gave us an arbitrarily large set of data with RDFS classes that we could use for load testing.

To service a high performing client-side application the triple store must be able to both return queries quickly and in large volume. We built a testing framework to benchmark the triple stores we were testing. This framework measured three areas, data loading time, large query runtime, and throughput of small queries.

Data load time was measured by converting the LUBM dataset into groups of 10 n3 formatted triples and then measuring both the time it took to load one million of those groups and the average load time per group of triples. These were loaded through the HTTP REST interfaces for each of the stores and the times were measured from the python script that was loading them.

Large queries were defined as a query that would be used to generate lists and as a starting point for more queries that would pull details out. To simulate this we ask for the URIs of all students, professors, classes, departments and universities. Once we have these we produce 1,000 smaller queries to ask for details about the individuals using increasingly complex queries.

There were three small queries that were run 1,000 times initially asking for the details of a single student, then asking for the details for all professors in one university, and finally asking for the details for all students taking classes under a specific professor. The average of these was used to produce the results.

2.3 Queries

This is a very simple query designed to test the minimum time that each triple store could return a single element.

Listing 1.1. Benchmark Query

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?name
WHERE { <STUDENT> ub:name ?name . }
```

The first query is to pull all of the information about a single student from many optional fields. In an application this would be common for anything displaying details about a record.

Listing 1.2. Query 1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?name ?advisor ?email ?telephone
WHERE { optional {?x ub:name ?name .}
        optional {?x ub:advisor ?advisor .}
        optional {?x ub:email ?email .}
        optional {?x ub:telephone ?telephone .}
        values ?x { <STUDENT> }
        }
```

This is a very simple query testing minimal time with the possibility of multiple rows.

Listing 1.3. Query 2

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X
WHERE
{?X ub:publicationAuthor <PAPER> .
  }
```

Another simple query testing time with a single join and multiple row return.

Listing 1.4. Query 3

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?student
WHERE {  ?student ub:takesCourse ?course .
         <PROFESSOR> ub:teacherOf ?course .
      }
```

A highly selective query with multiple joins to test inference as both faculty and student are inferred relationships.

Listing 1.5. Query 4

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X ?Y
WHERE
{?X rdf:type ub:Student .
  ?Y rdf:type ub:Faculty .
  ?X ub:advisor ?Y
  values ?Y { <PROFESSOR> }
 }
```

3 Results

3.1 Load Test

The results of the loading test shows a large difference between the materialized inference triple stores Sesame and Blazegraph and the query time inference model of Jena.

Table 1. Data load times

Store	Total	Mean	Stdev	Median
Jena	1776k	13	101	5
Blazegraph	39341k	309	429	12
Sesame	8414k	66	205	35

Measured in milliseconds

3.2 Queries

Table 2. Data load times

Store	Baseline Query	$\overline{x}$	Query 1	$\overline{x}$	Query 2	$\overline{x}$	Query 3	$\overline{x}$	Query 4	$\overline{x}$
Jena	7.78k	7	4396.23k	4413	7.9k	7	1545k	1678	9.17k	9
Blazegraph	37.36k	38	36.06k	36	37.11	37	32.7	36	34.91	38
Sesame	8.38k	8	253.97k	256	7.71k	7	7.48k	8	7.53k	8

Measured in milliseconds

4 Discussion

In client-side web applications response time is important for keeping users happy. Understanding why triple stores perform the way they do is helpful in choosing the correct one for any given project. We have demonstrated the large performance differences that can appear even in relatively small datasets between the different inference models.

References

1. Apache jena. `https://jena.apache.org/`.
2. Blazegraph. `http://www.blazegraph.com/bigdata`.
3. Centos. `https://www.centos.org/`.

4. Sesame. `http://rdf4j.org/`.
5. Tomcat. `http://tomcat.apache.org/`.
6. Virtualbox. `https://www.virtualbox.org/`.
7. Robert Battle and Edward Benson. Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST): Semantic Web and Web 2.0. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(1):61–69, 2008.
8. R.T. Fielding and R.N. Taylor. Principled design of the modern Web architecture. *Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium*, 2(2):115–150, 2000.
9. Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics*, 3(2-3):158–182, 2005.
10. Martin Voigt, Annett Mitschick, and Jonas Schulz. Yet another triple store benchmark? Practical experiences with real-world data. *CEUR Workshop Proceedings*, 912(Sda):85–94, 2012.