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 [1]. 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 [2].

2 Material and Methods

2.1 RDF Triple Stores

We have decided to focus on Apache Jena [3], Blazegraph [4], and Sesame [5] 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. [6]

Testing was performed on VirtualBox VM [7] with 2x 2.4 GHz 6-core Xeon processors and 8GB of ram. The operating system was CentOS 7 [8] and all of the stores were run through the Java application server Tomcat [9].

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) [10] 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

To begin with we need a query to determine the minimum time that each system will be able to return a query. Therefor we designed a query that would return a single object with a subject level lookup. STUDENT is replaced with the URI of a random student pulled from a list of all students.

Listing 1.1. Benchmark Query

```
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?name
WHERE { <STUDENT> ub:name ?name . }
```

Query 1 simulates returning the details about an individual where fields are not required. STUDENT is replaced with the URI of a random student pulled from a list of all students.

Listing 1.2. Query 1

```
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> }
        }
```

Query 2 returns all of the URIs of the authors on a given paper. This is testing return time where multiple rows will be returned. PAPER is replaced with a random URI of a paper.

Listing 1.3. Query 2

```
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?X
WHERE
{?X ub:publicationAuthor <PAPER> .
  }
```

Query 3 is an extension of query 2 except this has a join involved. PROFESSOR is replaced with a random URI of a professor.

Listing 1.4. Query 3

```
PREFIX ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT ?student
WHERE {  ?student ub:takesCourse ?course .
         <PROFESSOR> ub:teacherOf ?course .
      }
```

Query 4 is a highly selective query with multiple joins to test inference as both faculty and student are inferred RDF-level relationships. PROFESSOR is replaced with a random URI of a professor.

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

Table 1. Data load times for 10 rows

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

Measured in milliseconds

The results of the loading test shows a large difference between the three storage models of the triple stores. Blazegraph which has the largest number of indexes takes the longest to insert new data into. Sesame which materializes the inferences being made also takes longer than Jena which is doing inferences at query runtime.

3.2 Queries

Table 2. Mean query times for 1,000 queries

Store	Baseline Query $\overline{x}$	Query 1 $\overline{x}$	Query 2 $\overline{x}$	Query 3 $\overline{x}$	Query 4 $\overline{x}$
Jena	8	4413	7	1678	151
Blazegraph	38	36	37	36	38
Sesame	8	253	8	8	7

Measured in milliseconds

Jena's performance was highly volatile based on the query. The baseline speed was very fast as expected from the low number of indexes and the lack of inference in the baseline query. The OPTIONAL keyword in the first query is known to cause potential slowdowns and takes a heavy toll on Jena. Query 3 also has very poor performance but the reason for this is not clear. The inference required for query 4 slows it down in comparison to both Blazegraph and Sesame as it must calculate the inference at query time.

Blazegraph's large number of indexes and well established query optimizer result in extremely stable query results. There appears to be a 30ms processing time on anything Blazegraph is doing, this could be the query optimizer or another part of the system.

Sesame showed the best performance overall only having problems with the OPTIONAL keyword in the first query.

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 index and inference models.

References

1. R. Fielding and R. Taylor, "Principled design of the modern Web architecture," *Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium*, vol. 2, no. 2, pp. 115–150, 2000.
2. R. Battle and E. 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*, vol. 6, no. 1, pp. 61–69, 2008.
3. "Apache jena." `https://jena.apache.org/`.
4. "Blazegraph." `http://www.blazegraph.com/bigdata`.
5. "Sesame." `http://rdf4j.org/`.
6. M. Voigt, A. Mitschick, and J. Schulz, "Yet another triple store benchmark? Practical experiences with real-world data," *CEUR Workshop Proceedings*, vol. 912, no. Sda, pp. 85–94, 2012.
7. "Virtualbox." `https://www.virtualbox.org/`.
8. "Centos." `https://www.centos.org/`.
9. "Tomcat." `http://tomcat.apache.org/`.
10. Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems," *Web Semantics*, vol. 3, no. 2-3, pp. 158–182, 2005.