RDF Triple Store Performance For Applications

Joseph Utecht and Mathias Brochhausen PhD

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

Abstract. Although many RDF triple store performance measure have been published, there appears to be a gap when it comes to their use as the primary storage for applications. The performance needs for this use case differ from the triple stores more traditional use of reasoning and inference over large data sets. The primary objective of this research is to determine the feasibility of modern RDF triple stores to be used as the primary storage for a responsive application. To make this determination we measured the load time of just over one million triples of synthetic data and then ran various queries to emulate the types of demands this use would create. The resulting time measurements showed a difference in load times between the different stores and a large range of results in query performance. The lower end results are fast enough to compete with traditional relational databases while the results on the high end would not be suitable for this use case. The varied performance showed that while it was certainly possible to use triple stores in this way the developer must be careful with query design and understand some of the performance implications of different storage models that various stores offer.

1 Introduction

Trauma Centers and trauma systems have been proven to increase the positive outcome in trauma care [1–3]. However, comparison of trauma centers and of trauma systems and the patient outcome created by them is obstructed by a lack of comparability regarding the organizational components and the terms used to refer to organizational structures and roles. The NIH-funded project *Comparative Assessment For Environments of trauma care* (CAFE) (1R01GM111324-01A1) aims to develop a semantically driven, web-based IT framework to collect data about the organizational structure of trauma centers and trauma systems and to allow comparison of different structures for either trauma centers or trauma systems. To achieve that CAFE will create a knowledge resource based on an RDF triple store to store data about the organizational structure of both trauma centers and trauma systems. Knowledge about types of entities relevant to the domain will be provided by an ontology coded in Web Ontology Language 2 (OWL2) [1]. The system will allow users to enter data about the organizational

[1] `http://www.w3.org/TR/owl2-primer/`

structure of a trauma center or a trauma system. The information will be captured in RDF and the progress in entering data will be perceivable to user by the growing graphical representation of the organizational structure. In the next step the user can request a comparison of their organization with canonical organizations of the same type. The data about those organizations will be stored in the triple store. The system will run a set of queries that retrieve data that allows representing the same organizational units described by the users data and their interrelations. The user will be presented with two graphical representations: their own organization and a canonical organization of the same type.

RDF triple stores are not generally used as the primary data storage means for an application. Because of this, most studies tend to focus on the performance of triple stores as it relates to reasoning and complex queries over large data sets. The CAFE project requires the ability to insert, reason and query data in as close to real time as is possible so the user experience does not suffer. We would like to attempt to use an RDF triple store to perform all of these duties. In this paper we will attempt to measure the performance of various RDF triple stores as they would be used to drive an application.

2 Material and Methods

2.1 RDF Triple Stores

We have decided to focus on Apache Jena [2], Blazegraph [3], 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. [4]

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

Apache Jena uses a Java based front-end called Fuseki 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. Blazegraph uses up to six indexes to store the data and has a mature query optimizer [5]. If inference is enabled Blazegraph will materialize the inferences with a context which allows them to be removed if the RDF classes are modified.

[2] https://jena.apache.org/
[3] http://www.blazegraph.com/bigdata
[4] http://rdf4j.org/
[5] https://www.virtualbox.org/
[6] https://www.centos.org/
[7] http://tomcat.apache.org/

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 store and RDFS inference enabled. Sesame allows arbitrary indexes to be created when the graph is initialized but we kept the default two. Like Blazegraph, Sesame materializes RDF inference when data is added.

2.2 Testing Method

We used Lehigh University Benchmark (LUBM) [6] generated data to test performance and capability of the triple stores. LUBM is a way to produce a synthetic dataset of arbitrary size. The data produced represents people and activities in a university setting. We did not use the default testing queries with this dataset as they were more designed to measure the performance of OWL inference models.

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 two areas, data loading time 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 the time it took to load one million of those groups. 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.

SPARQL queries were used to generate lists of URIs as a starting point for the small queries with which we were testing query runtime. To this end we queried for the URIs of all students, professors, classes, departments and universities. Once we had these we ran the five queries below 1,000 times through the HTTP REST interface measuring the response time with a python script.

2.3 Queries

To begin, a query to determine the baseline time that each system will be able to return a query is needed. Therefor we designed a query that would return a single object with a subject level lookup, which should be the fastest possible lookup operation for any store. 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 to be present. The *optional* keyword can potentially cause long runtimes so this is a naive approach to this query. In the query 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 ?author
WHERE
{ ?author ub:publicationAuthor <PAPER> . }
```

Query 3 also returns multiple rows and also 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 to test inference as both ub:Faculty and ub:Student are inferred RDF-level 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 ?student ?faculty
WHERE
{ ?student rdf:type ub:Student .
  ?faculty rdf:type ub:Faculty .
  ?student ub:advisor ?faculty
  values ?faculty { <PROFESSOR> } }
```

3 Results

The results of the loading test show 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 and Jena have similar numbers

Table 1. Data load times for 10 triples

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

Measured in milliseconds

Table 2. Mean runtime 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

of indexes, however Sesame is materializing the inferences and thus has slower load times.

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 however 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 Conclusion

In 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.

Acknowledgement This is where we thank the NIH XXXXXX

References

1. D. Demetriades, M. Martin, A. Salim, P. Rhee, C. Brown, J. Doucet, and L. Chan, "Relationship between American College of Surgeons trauma center designation and mortality in patients with severe trauma (injury severity score > 15)," *J. Am. Coll. Surg.*, vol. 202, pp. 212–215, Feb 2006.
2. T. L. Sanddal, T. J. Esposito, J. R. Whitney, D. Hartford, P. P. Taillac, N. C. Mann, and N. D. Sanddal, "Analysis of preventable trauma deaths and opportunities for trauma care improvement in Utah," *J Trauma*, vol. 70, pp. 970–977, Apr 2011.
3. R. J. Winchell, N. Sanddal, J. Ball, H. Michaels, C. R. Kaufmann, R. Gupta, T. J. Esposito, and H. Subacius, "A reassessment of the impact of trauma systems consultation on regional trauma system development," *J Trauma Acute Care Surg*, vol. 78, pp. 1102–1110, Jun 2015.
4. 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.
5. G. B. Olivier Curé, *RDF Database Systems: Triples Storage and SPARQL Query Processing.* Morgan Kaufmann, 1st ed., December 2014.
6. 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.