Scalability Report on Triple Store Applications

Ryan Lee
July 14, 2004

`ryanlee@w3.org`
`http://simile.mit.edu/`

Abstract

This report examines a set of open source triple store systems suitable for The SIMILE Project's browser-like applications. Measurements on performance within a common hardware, software, and dataset environment grant insight on which systems hold the most promise for acting as large, remote backing stores for SIMILE's future requirements.

The SIMILE Project (Semantic Interoperability of Metadata In like and Unlike Environments) is a joint research project between the World Wide Web Consortium (W3C), Hewlett-Packard Labs (HP), the Massachusetts Institute of Technology / Computer Science and Artificial Intelligence Laboratory (MIT / CSAIL), and MIT Libraries. Funding is provided by HP.

Contents

1 Introduction

The goal of the Semantic Web is to ”provide a common framework that allows data to be shared and reused across application, enterprise, and community boundaries.” [15] While much of the research on the Semantic Web to date has been in specifying the mechanics of its operation, the Semantic Interoperability of Metadata In like and unLike Environments (SIMILE) project seeks to apply Semantic Web principles to real world situations. The SIMILE project is a broad experiment to test the present state of Semantic Web technology within the real-world library domain and the framework of the DSpace digital library system [7].

This report focuses on the performance of existing data store systems built specifically for the Semantic Web. Our intended audience is expected to have a firm background in RDF and its applications. Those looking for an introduction to the Semantic Web may want to start elsewhere, perhaps with the W3C Semantic Web Activity website found at <http://www.w3.org/2001/sw/>.

2 Problem Statement

The applications SIMILE is currently building deal primarily with browsing large data stores. For our purposes, a data store is geared towards storing and returning RDF triples in response to queries. Inferencing was not an important consideration though the existence of such features is noted. The metrics we are most interested in are the speeds of loading data into the store, loading data into the application, configuring the application according to the data, and queries with large expected results; not all steps are applicable to each system we examined.

Future SIMILE applications may also be concerned with the traditional database transactions of inserting, updating, and deleting data, but those transactions are beyond the scope of this report.

2.1 Related Work

The Semantic Web Advanced Development for Europe (SWAD-E) project published a survey [4] of free software / open source RDF stores available in 2002. The main concerns of their survey involved the licensing, interfaces, and language used in writing each system.

3 Testing Environment

Test results are presented from experiments conducted in a consistent hardware environment using the same set of data within the same application.

3.1 Hardware Environment

The test machine is an Apple Powerbook G4 1.25GHz with 1GB of memory. The test results should be considered in relation to one another since no one is going to seriously run an applications for public consumption off of a laptop.

3.2 Dataset

The dataset is a subset of ARTstor [3] art metadata in addition to a subset of the MIT OpenCourseWare (OCW) [12] metadata, both originally given to SIMILE in XML format. A small number of ontologies, some of them custom made by SIMILE for the ARTstor and OCW datasets, were used in composing the data in RDF. The raw size is 17.2 MB in RDF/XML format (170,244 triples); after some application-specific inferencing, including basic OWL inferencing, the dataset size is 27.4 MB in RDF/XML format (279,337 triples). In all tests, we used the post-inferencing dataset.

These datasets were used because of their availability to us; licensing restrictions prevent us from sharing them in their entirety. The information represented within the data is generally representative of what can be found in the library domain.

At the graph level, the data structure is fairly flat; that is, there are several nodes, some sharing property-object combinations, but most nodes are not connected to one another. It is comparable to a couple of very large tables with few references in a relational database.

3.3 Application Behavior

Our testing application, named Longwell, is a faceted RDF browser. Adding support for a new store to Longwell involves implementing one interface, making it a simple task to add a new store and to configure which store the application uses. There are two main phases of operation in Longwell: configuration and user interaction. A pre-configuration phase for loading data into a store is also considered for in-memory models, where loading is handled by the application, and for local models, where loading is not handled by the application and depends on the store's input mechanism.

The configuration phase involves setting up an in-memory RDF model cache, used in *Hybrid* and network models, and other cache-like performance enhancements. A separate display configuration phase follows. When both configuring phases are complete, the web application is made available to users for browsing the loaded data via their user-agent of choice. For our tests, the user load was restricted to a single user.

For each system examined, the following statistics are given:

Read in RDF data	Time to load data into the application or local store, if applicable
Configure browser	Time to initialize caching and optimizations
Display setup	Time to set up proper display
Load page	Time to load first page of application, by far the largest

4 Stores and Results

The following applications were found to be in a state ready or nearly ready for basic testing with our application. Each of the applications is treated with a brief description of its authors and intended use, underlying technology, additional features unused by our application, interaction method with Longwell, and the respective test results.

Comprehensive results comparing all the stores we examined are in the following section, Section 5.

4.1 Jena

Hewlett-Packard Labs' SWAD-E team maintains Jena, "a Java framework for building Semantic Web applications," [9] available as open source software under a BSD license. Jena implements APIs for dealing with Semantic Web building blocks such as RDF and OWL; our Longwell application is built upon Jena.

Jena's fundamental class for users is the `Model`, an API for dealing with a set of RDF triples. A `Model` can be created from the filesystem or from a remote file. Using JDBC, it can also be tied to an existing RDBMS such as MySQL or PostgreSQL, the two open source RDBMS's we tested.

There are three variants of dealing with Jena's `Model` presented in the results. The `JenaLocalModel` behaves as if the entire set of triples were in-memory. While actually storing everything in-memory cannot be a serious method for storing extremely large volumes of data, it is a useful benchmark to measure against and a potentially useful way to cache data obtained from remote stores.

The `RDQLLocalModel` naively queries a `Model` using the RDQL query language instead of Jena's `Model` API. Very little is cached for the users' sake, so all user interaction through the web application ends up relying on accessing the store, and through non-optimal queries at that.

The `RDQLHybridModel` uses RDQL queries to initialize an in-memory cache from a store. All user interaction after initialization uses Jena's `Model` API on the cached data and does not require any communication with the store. This is a very simple caching strategy that does not require any replacement strategies; we anticipate this will change as our datasets grow.

4.1.1 Additional Features

Jena contains a rich set of features for dealing with RDF including an inferencing engine, the ability to reason using custom rules, and OWL-based ontology processing.

4.1.2 Filesystem

JenaLocalModel

Read in RDF data	66979 ms
Configure browser	247 ms
Display setup	14 ms
Load page	1995 ms

RDQLLocalModel

Read in RDF data	67143 ms
Configure browser	2 ms
Display setup	51 ms
Load page	34885 ms

RDQLHybridModel

Read in RDF data	67503 ms
Configure browser	19306 ms
Display setup	42 ms
Load page	2069 ms

4.1.3 MySQL 3.x

MySQL versions 3.23 and 4.0.17 were both used in testing. Loading the dataset into MySQL 3 took 667138 ms.

JenaLocalModel

Configure browser	3729 ms
Display setup	208 ms
Load page	69013 ms

RDQLLocalModel

Configure browser	4 ms
Display setup	78 ms
Load page	85767 ms

RDQLHybridModel

Configure browser	50693 ms
Display setup	52 ms
Load page	1638 ms

4.1.4 MySQL 4.x

Loading took 844257 ms in MySQL 4.

JenaLocalModel

Configure browser	5266 ms
Display setup	242 ms
Load page	81039 ms

RDQLLocalModel

Configure browser	2 ms
Display setup	126 ms
Load page	97682 ms

RDQLHybridModel

Configure browser	30542 ms
Display setup	116 ms
Load page	1843 ms

4.1.5 PostgreSQL

With PostgreSQL version 7.4, loading the dataset into the database took 971784 ms.

JenaLocalModel

Configure browser	6194 ms
Display setup	298 ms
Load page	115791 ms

RDQLLocalModel

Configure browser	3 ms
Display setup	95 ms
Load page	111246 ms

RDQLHybridModel

Configure browser	27526 ms
Display setup	54 ms
Load page	1950 ms

4.2 Joseki

The SWAD-E group at HP Labs also maintains Joseki, a web application ”for publishing RDF models on the web.” [10] Joseki is also available under a BSD license. It is built on Jena and, via its flexible configuration, allows a `Model`, represented as a set of files or within a database, to be made available on a specified URL and queried using a number of languages. Results can be returned as

RDF/XML, RDF/N3, or NTriples. The query languages, result formats, and model sources can be extended to produce new alternatives tailored to the user's needs.

Tests with different result formats and different stores all using the RDQL query language are presented below. Note that our application and the Joseki application were running on the same machine, so during the configuration phase, network communication was at an optimum (no collisions), but processor resources were divided. In a real setup, the Joseki application would more likely be running on a remote machine. As in the *Hybrid* style models, results from Joseki queries are cached in-memory during the configuration phase, and no further communication with the remote store is necessary after initialization is complete.

4.2.1 Additional Features

Joseki's other query mechanisms include fetching the entire model, fetching only the direct relations to a particular node, and a subject-predicate-object query language similar to Jena's API.

Some small extensions were added through Joseki's extension mechanism to enable Joskei and Kowari to work together (see the Kowari section below). This turned out to be an easy and quick operation.

4.2.2 RDF/XML Response Format

Joseki's default response format is the normal RDF/XML serialization.

Filesystem

Configure browser	172911 ms
Display setup	5347 ms
Load page	1752 ms

MySQL

Configure browser	209485 ms
Display setup	5281 ms
Load page	1508 ms

PostgreSQL

Configure browser	266691 ms
Display setup	7134 ms
Load page	1490 ms

4.2.3 RDF/N3 Response Format

The informal RDF/N3 serialization tends to be more concise than the same graph expressed in RDF/XML; it was hoped the savings in bandwidth would lead to better performance.

Filesystem

Configure browser	181657 ms
Display setup	5044 ms
Load page	1510 ms

MySQL

Configure browser	217934 ms
Display setup	5609 ms
Load page	1725 ms

PostgreSQL

Configure browser	193682 ms
Display setup	6910 ms
Load page	1803 ms

4.3 Kowari

Tucana Technologies [17] provides an open source version of its commercial Tucana Knowledge Server called Kowari [11]. Kowari is an entirely Java based transactional, permanent triple store available under the Mozilla Public License. It does not rely on an external RDBMS to provide the actual store, implementing its own transactional database instead. Several different methods of communication with a Kowari store are available, including RDQL via its implementation of the Jena `Model`, though a custom language named iTQL (Interactive Tucana Query Language) is better documented. iTQL allows some flexibility not in RDQL such as result set size limits. We moved from version 1.0.1 to 1.0.3 during testing.

4.3.1 Additional Features

Kowari includes a Lucene index of stored data for full-text searching. Kowari offers connections through SOAP and RMI as well as implementations of interfaces from Jena and JRDF. Management of the stores is done through a web application. Also included is a feature called Descriptors which appears to be similar to an XSLT pipeline, but we did not make use of this feature.

4.3.2 In-Memory

It is possible to create a Kowari store within our Longwell application, but, as above, the in-memory store is not one we are seriously evaluating as a long-term solution. As a separate implementation of Jena's `Model` interface, it was interesting to see how it compared, if only because implementations other than Jena's are somewhat rare.

Read in RDF data	177778 ms
Configure browser	2166 ms
Display setup	95 ms
Load page	107013 ms

4.3.3 Network

Kowari 1.0.3 provides several interfaces for querying data from a remote store. Loading data from the filesystem into Kowari took 139092 ms. The following test was done with the naive query strategy using iTQL over Java's Remote Method Invocation (RMI) protocol. As the querying strategy is poor and used for every part of every user page view in Longwell, the amount of time it took to load the first page was beyond the author's patience. This is not a serious strategy, only intended to be a benchmark. The subsequent model used iTQL over RMI to generate an in-memory cache. The end result for users was substantially better.

iTQL over RMI

Configure browser	9204 ms
Display setup	411 ms
Load page	$> 10^7$ ms

iTQL with caching over RMI

Configure browser	80304 ms
Display setup	479 ms
Load page	1670 ms

Because Kowari implements Jena interfaces, it was a good candidate for use as an alternate source in Joseki.

Kowari+Joseki, RDQL over HTTP, RDF/XML response

Configure browser	194442 ms
Display setup	9304 ms
Load page	1602 ms

Kowari+Joseki, RDQL over HTTP, RDF/N3 response

Configure browser	205763 ms
Display setup	10074 ms
Load page	1661 ms

4.4 3store

3store [1] is maintained by a development team at The University of Southampton and used within the Advanced Knowledge Technologies (AKT) project. 3store ”is a core C library that uses MySQL to store its raw RDF data and caches” [2] and relies on Redland [14], an RDF interface project grounded in its C library, developed by Dave Beckett of Bristol University's Institute for Learning and Research Technology. 3store is available under the GPL, and Redland under the LPGL or MPL licenses.

During our testing, 3store moved versions from 2.2.14 to 2.2.17, in part due to developments discovered by our testing. Contributions from the 3store developers enabled our tests to proceed. As of the release of this report, note that 3store works best with MySQL 3.x versions, not MySQL 4.x.

We wrote a small Java class using SAX to deal with the XML results 3store returns when queried via RDQL over HTTP to its Apache module. Again, note that the network delay here is at a minimum while processing resources were split between Apache, MySQL, and Longwell. The results were cached in an in-memory store for browsing.

4.4.1 Additional Features

3store can infer RDF and RDFS entailments and can also communicate using the OKBC protocol through an Apache module.

4.4.2 Network

Loading data into 3store took 213088 ms.

Configure browser	38440 ms
Display setup	2371 ms
Load page	1908 ms

4.5 Sesame

"Sesame is an open source RDF database with support for RDF Schema inferencing and querying." [16] Now at version 1.0.2, Sesame is distributed by Aduna BV, NLnet Foundation, and volunteers under the GPL license. Written in Java, Sesame is intended to be run as a web application; the documentation seems to indicate using Tomcat as the servlet container is easiest, so we used Tomcat 5.0.25.

Much like Jena, Sesame can use open source RDBMS's MySQL and PostgreSQL in addition to its in-memory database. All three options were tested.

4.5.1 Additional Features

It is possible to run Sesame as a standalone repository without the web application portion, but this wasn't tested due to time constraints and a lack of architectural interest (since our project does not intend to constrain repositories to local operation in the long term). Sesame also supports custom inferencing and can perform RDFS entailment.

4.5.2 Network

Precise loading times were not gathered, though the order of speed was loading into memory, then MySQL, and finally PostgreSQL. Queries were written in RDQL and sent over HTTP. The response returned the answer in Sesame's XML format and parsed in our application by Sesame's client libraries. The result was an in-memory cache used for browsing.

In-Memory

Configure browser	20488 ms
Display setup	1428 ms
Load page	1745 ms

MySQL 3

Configure browser	40865 ms
Display setup	2186 ms
Load page	1887 ms

PostgreSQL

Configure browser	66970 ms
Display setup	4378 ms
Load page	2047 ms

4.6 Others

We are aware of other existing open source RDF stores, particularly asemantics' RDFStore and MIT CSAIL's Cholesterol, a component of the Haystack framework, but we were unable to do a timely evaluation of their work due to programming language barriers (our application is written entirely in Java, and our time constraints did not allow us to write full Java APIs or RDQL interfaces for those projects that did not already implement their own).

We are interested in other options and certainly cannot claim this report to be a comprehensive survey, but the more work required to glue our system and a store together, the less likely we will be able to do it.

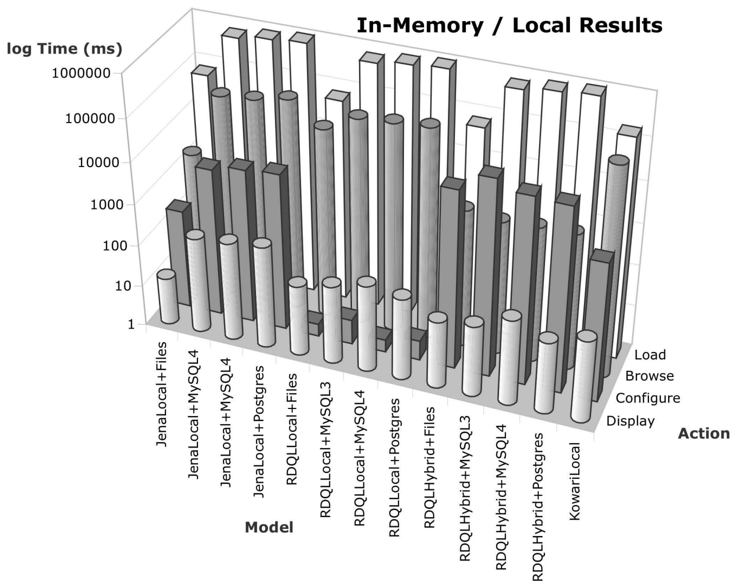
5 Comprehensive Results

5.1 In-Memory / Local

Given our architectural desire to place data stores physically separate from our application, the in-memory or otherwise local interaction models will very likely not be used as full stores, despite their generally better performance. However, some combination of in-memory or local store may be useful for caching purposes. Combined results for all non-remote stores are given in table 1.

Model	Load	Configure	Display	Browse
JenaLocalModel (Files)	66979	247	14	1995
JenaLocalModel (MySQL 3)	667138	3729	208	69013
JenaLocalModel (MySQL 4)	844257	5266	242	81039
JenaLocalModel (Postgres)	971784	6194	298	115791
RDQLLocalModel (Files)	67143	2	51	34885
RDQLLocalModel (MySQL 3)	667138	4	78	85767
RDQLLocalModel (MySQL 4)	844257	2	126	97682
RDQLLocalModel (Postgres)	971784	3	95	111246
RDQLHybridModel (Files)	67503	19306	42	2069
RDQLHybridModel (MySQL 3)	667138	50693	52	1638
RDQLHybridModel (MySQL 4)	844257	30542	116	1843
RDQLHybridModel (Postgres)	971784	27526	54	1950
KowariLocalModel	177778	2166	95	107013

Table 1: In-Memory / Local

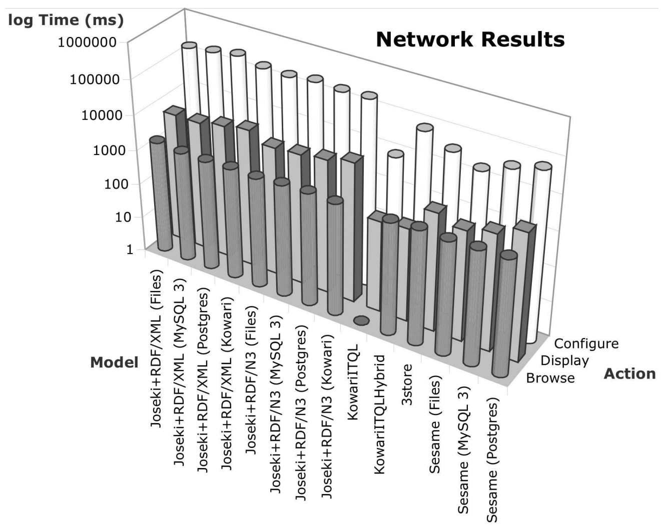


5.2 Network

Combined results for all remote stores are shown in table 2.

Model	Configure	Display	Browse
Joseki+RDF/XML (Files)	172911	5347	1752
Joseki+RDF/XML (MySQL 3)	209485	5281	1508
Joseki+RDF/XML (Postgres)	266691	7134	1490
Joseki+RDF/XML (Kowari)	194442	9304	1602
Joseki+RDF/N3 (Files)	181657	5044	1510
Joseki+RDF/N3 (MySQL 3)	217934	5609	1725
Joseki+RDF/N3 (Postgres)	193682	6910	1803
Joseki+RDF/N3 (Kowari)	205763	10074	1661
KowariITQLModel	9204	411	$> 10^7$
KowariITQLHybridModel	80304	479	1670
3store	38440	2371	1908
Sesame (Files)	20488	1428	1745
Sesame (MySQL 3)	40865	2186	1887
Sesame (Postgres)	66970	4378	2047

Table 2: Network



6 Conclusion

Drawing conclusions about remotely accessible stores is more pertinent to our project requirements.

In passing, it seems MySQL 3 performs the most quickly in general as a Jena store, and Kowari shows some great promise with its order of magnitude less time for configuration and its speed of loading data into the store.

Browsing and configuration times were the most pertinent figures to our future work. We don't believe the browsing times are really significant beyond the second granularity, so by that metric, it appears models with a performance between one and two seconds are potentially worth pursuing. All of our network models with caching appear to fall in that range, which is perhaps not a surprise since all of them implement caching in approximately the same fashion.

This leaves configuration time as the more interesting metric - how fast does a store return its results for creating the in-memory cache? For network models, the fastest were 3store and Sesame with files, though using files for the remote store is akin to using an in-memory model for our application, meaning it probably is not feasible for extremely large stores. So 3store and Sesame using MySQL 3 appear to be our best choices.

The non-Joseki-based remote stores are approximately an order of magnitude faster at cache initialization. Perhaps this should be expected because the focus of the Jena and Joseki projects has been more on doing things correctly instead of doing them as quickly as possible.

These single-run figures from a laptop seem to indicate that at this scale, Sesame and 3store are the most worthwhile for exploring as remotely accessible stores for even larger datasets. Given that future caching strategies will probably require more advanced techniques than the current simple one of grabbing everything and sticking it all in memory and never replacing anything, a fast turnaround from the store to our application is essential.

6.1 Future Work

There are other stores worth examining, and there is more data for our project to consume. We hope to continue in the same vein as this initial foray in the future with another report on larger scale store performance, perhaps with a broader selection of systems to compare against.

Appendix: Acronyms and Terms

Apache Shorthand for the Apache Foundation's httpd web server

API Application Programming Interface

CSAIL MIT Computer Science and Artificial Intelligence Laboratory

HTTP HyperText Transfer Protocol

JDBC Database connection standard for Java

Lucene Text-searching software provided by the Apache Foundation

N3 An informal shorthand serialization of RDF

NTriple An informal serialization of RDF, similar to but distinct from N3

OKBC Open Knowledge Base Connectivity

OWL Web Ontology Language

RDF Resource Description Framework

RDFS RDF Schema

RDQL RDF Data Query Language

RMI Remote Method Invocation

SAX Simple API for XML

SIMILE Semantic Interoperability of Metadata in Like and unLike Environments

SWAD-E Semantic Web Advanced Development for Europe

XML Extensible Markup Language

References

[1] 3store. <http://sourceforge.net/projects/threestore/>

[2] AKT - Technologies - 3store from the University of Southampton. <http://www.aktors.org/technologies/3store/>

[3] ARTstor. <http://www.artstor.org/>

[4] D. Beckett. SWAD-Europe: Scalability and storage: Survey of free software / open source RDF storage systems, 2002. <http://www.w3.org/2001/sw/Europe/reports/rdf_scalable_storage_report/>

[5] D. Beckett. SWAD-Europe: Tools for Semantic Web Scalability and Storage: Implementation report on scalable Free Software /Open Source RDF storage system, 2003. <http://www.w3.org/2001/sw/Europe/reports/rdf_scalable_storage_impl/>

[6] D. Beckett, J. Grant. SWAD-Europe: Mapping Semantic Web Data with RDBMSes, 2003. <http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/>

[7] DSpace. <http://www.dspace.org/>

[8] S. Harris, N. Gibbins. 3store: Efficient Bulk RDF Storage, 2003. <http://km.aifb.uni-karlsruhe.de/ws/psss03/proceedings/harris-et-al.pdf>

[9] Jena. <http://jena.sourceforge.net/>

[10] Joseki. <http://www.joseki.org/>

[11] Kowari. <http://www.kowari.org/>

[12] MIT OpenCourseWare. <http://ocw.mit.edu/>

[13] E. Prud'hommeaux, B. Grosof. RDF Query Survey, 2001. <http://www.w3.org/2001/11/13-RDF-Query-Rules/>

[14] Redland RDF Application Framework. <http://www.redland.opensource.ac.uk/>

[15] W3C Semantic Web Activity. <http://www.w3.org/2001/sw/>

[16] Sesame. <http://www.openrdf.org/>

[17] Tucana Technologies, Inc. <http://www.tucanatech.com/>