PNNL-17237

# Provenance Store Evaluation

P Paulson
T Gibson
K Schuchardt
E Stephan

March 2008

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

PNNL-17237

# Provenance Store Evaluation

P Paulson
T Gibson
K Schuchardt
E Stephan

March 2008

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99352

## Summary

Requirements for the provenance store and access API are developed. Existing RDF stores and APIs are evaluated against the requirements and performance benchmarks.
The team's conclusion is to use MySQL as a database backend, with a possible move to Oracle in the near-term future. Both Jena and Sesame's APIs will be supported, but new code will use the Jena API.

# Contents

# Figures

# Tables

# 1.0 Functional and Performance Requirements

We want to support the provenance steps described by ,   specify 4 phases in the provenance lifecycle: creation, recording, querying, and managing. The RDF based Provenance store should support each of the phases.

## 1.1 Provenance creation, recording, and querying

The provenance store should provide APIs or web services to allow users to specify new provenance information; it must also support the storage of a large amount of provenance information. In addition, the store must support queries for all provenance related to some data instance. This may require substantial time to transitively find all information related to a data item.

## 1.2 Provenance management

The system needs to provide tools to support standard data-management tasks. These tasks may include backups and restore, journaling and crash-recovery, purging, data-reorganization, and storage optimization.

### 1.2.1 Data security, Reliability, Availability, and Fault-toleranance

Because the projected customers require global access, the system should be capable of 24X7 operations, which requires online data backup and recovery. Failure of the provenance store should not prevent the execution of client processes; ideally, local provenance stores can provide temporary storage in case of network or server failure. Fail-over processing should be provided.

### 1.2.2 Capacity, Scalability, and Extensibility

Provenance assertions will be generated for every intermediate result generated by the system. We're assuming that the result sets will have high granularity—that is, there will not be provenance associated with each item in a dataset, but the data set as a whole. Historical provenance records will be kept for a window, but a purge process can be created to remove records which are unused.

We're assuming this implies that the capacity must be at least on the order of millions of data-items. Potentially, the system should be able to scale to the order or trillions of data-items.

### 1.2.3 Access and Integrity

It is assumed that access to actual data-items will be controlled by client systems. Although not envisioned for prototype systems, user-level access control should be supported for provenance records.

The system should support ACID Transaction support and journaling. Once a client receives confirmation of a commit, all p-assertions submitted as a transaction are guaranteed to persist in the store; if confirmation of a commit is not sent, the persisted store will not reflect any of the processing steps taken as part of the transaction.

## 1.3 Speed and Latency Requirements

For provenance creation, recording, and querying, the system should not cause significant delays to client programs; as much as possible, any additional processing time should be deterministic.

# 2.0 Evaluation of RDF Stores for Provenance Recording

Using the requirements as a guideline, we can come up with a set of dimensions that can be used in evaluation of potential RDF Storage systems. The RDF stores under consideration are composed of several components, some of which are interoperable between systems. A preliminary decomposition identifies 3 system components—the storage engine (such as MySQL tables or proprietary file system), API (such as Jena or OpenRdf), and the server software (Joseki is one example). Many of the dimensions described below apply to only 1 component. In addition, some capabilities apply only to the query languages the system's API and server software support. Table 2.1 outlines criteria to evaluate system components. Table 2.2 gives criteria that are only applicable to the server component.

**Table 2.1.** Criteria applied to multiple system components

| *Criterium* | *Requirements* | *Components* | *Description* |
|---|---|---|---|
| Web Interface | 1.1 | Server, API | Ability to support provenance creation, recording, and querying through web services |
| Query languages | 1.1 | Server, API | The query language supported, the expressiveness of the query language, and the acceptance/support of the query language. |
| Transitivity | 1.1 (querying) | Server, API | Some queries will involve all items that form the provenance of a particular item. Since existing query languages such as SPARQL do not support transitivity, this will require walking back through provenance chains. How this should be implemented – batch processing, interactive processing, or a built-in reasoner—is an open problem. |
| Reification | 1.2 | Storage,API | From the user's point of view, the RDF store will contain statements, or triples. For data-management purposes, it may be useful to store metadata about those statements. A pure RDF solution would create *reified* statements in a separate RDF store—properties, such as last access time, required access privileges, and other house-keeping details, about the target statements could then be stored. Some RDF stores, however, might supply low level access to the statements inside the store, supporting this functionality. This functionally will help support data backups, increase capacity and scalability through support of purge operations, and support data access by storing access information with triples. |
| Community Support | *all…* | *all…* | Ongoing commercial acceptance and community support will ensure development of new management tools and integration with new technologies. |
| Speed and Latency | 1.3 | Storage, Server | Benchmarks should be developed to evaluate stores in terms of the performance of insertion of new provenance records using a web interface. Insertion should be measured into an existing store a large number of triples in it (say 15M?) and the performance of queries accessing provenance information using a web interface. In addition, a combined benchmark should be designed to perform queries and insertions simultaneously to evaluate potential locking problems. |
| Capacity | 1.2.2 | Storage, Server | A high-capacity benchmark should be created to evaluate volume capacity of RDF stores |

**Table 2.2.** Additional criteria for storage component

| *Criterium* | *Requirements* | *Description* |
|---|---|---|
| Capacity | 1.2.2 | Ability to store a large number of triples |
| Multi-volume | 1.2.2 | Systems that support multi-volume RDF Stores will simplify high capacity data-storage and scale-ability. |
| Data Management Tools | 1.2 | Data management tools to support maintenance of the RDF Store. |
| Online Backups | 1.2.1 | To support Requirement 1.2.1, the RDF store should ideally support online backups, although this could probably be handled procedurally with most the systems being evaluated. |
| Shadowing/Replication | 1.2.1 | In order to provide robustness and fail-safe operations, systems that support database shadowing with automatic replication are desirable. This will allow automatic fail-over to be implemented so that updates can be made on any one of several servers, with all servers kept in sync by the data management system when they come back online. |
| Access control, Store level | 1.2.3 | Does the RDF store enforce user-level access control on the RDF Store? This would allow different levels of provenance to be stored in different RDF stores, controlling access to provenance information (design of this would still be difficult – which statements go into which store?). Supports requirement 1.2.3. |
| Access Control, View level | 1.2.3 | An RDF Store that supports access control on views within the underlying RDBMS could offer flexibility on access control. (But probably not much – still have to decide which view to use….) |
| Transaction Support | 1.2.1, 1.2.3 | Does the RDF Store support transactions with commit and rollback and journaling to protect against hardware failures? |

# 3.0 Overview of Candidate RDF Stores

## 3.1 APIs

### 3.1.1 Jena

Jena (http://jena.sourceforge.net/) provides

1. An API for manipulating RDF graphs
2. Support for multiple reasoning engines – OWL-DL (through Pellet), OWL-Lite, and RDF Schema
3. Support for multiple back-end storage systems, including
    a. native support for in-memory graphs
    b. RDBMS table storage, implemented for Oracle, SQL Server, MySQL, and Postgres
4. Support for the SPARQL query language
5. Server software (Joseki) that supports the SPARQL query language

Web sources (http://esw.w3.org/topic/LargeTripleStores) indicate installations handling 200M triples using Postgres as the storage engine.

### 3.1.2 Sesame

Sesame provides

1. An API for manipulating RDF graphs
2. Server software that supports the SeRQL query language
3. Support for a proprietary, file-based storage system
4. Reasoning over RDF Schema

Version 1.0 of Sesame also supported RDBMS table-based storage, but this has not yet been implemented for version 2.0. I was unable to decipher the documentation for version 1.0 support. Web sources indicate fair performance with systems of up to 70M triples.

### 3.1.3 Mulgara/Kowari

Mulgara is an open-source fork of Kowari. The marketing literature indicates that the design is meant to be scaleable to extremely large graphs. The system uses memory-mapped files and is tailored to 64-bit systems. Web sources indicates good performance with stores of 160M triples (http://esw.w3.org/topic/LargeTripleStores)

Mulgara provides:

1. A server supporting the Itql query/update language
2. A proprietary storage backend

### 3.1.4 3Store

Web sources (http://esw.w3.org/topic/LargeTripleStores) indicate successful applications handling 100M triples. This product provides a C language library. Untested since compiling on cygwin didn't go very smoothly – probably best on a Unix or Macintosh, but we're currently benchmarking on a windows machine. Uses MySql as backend.

Provides
1. Sparql Support
2. Store-level access control
3. Uses MySQL

### 3.1.5 RDF Gateway

Web sources(http://esw.w3.org/topic/LargeTripleStores) indicate installations handling 262M triples.
1. Commercial, free for evaluation.
2. RDF Gateway is a complete application and web server that manages a built-in RDF Store.
3. A server supporting the proprietary RDFQL query language. It looks like SPARQL is also supported
4. A proprietary storage backend
5. Access control using NT user and groups
6. Transaction Support
7. 'context' for statement could possibly support statement reification
8. content-level access control

We were unable to determine if on-line backups are supported.

Documentation for this product was too incomplete to allow me to easily code benchmarks for it, although it appears feasible.

### 3.1.6 BigOWLIM

One source claimed that this system handled 1.06B statements – adding more statements through OWL inferencing, with a load time was approximately 70 hrs. (http://esw.w3.org/topic/LargeTripleStores).

BigOWLIM is a reasoning and persistence implementation for the Sesame framework. It uses a proprietary disk storage system and implements RDFS and limited OWL entailment (does not support OWL-Lite).

BigOWLIM is not open source—it was not tested due to licensing limitations.

### 3.1.7 Garlik

Handles 1.7B triples, according to http://esw.w3.org/topic/LargeTripleStores. www.garlik.com describes a data-privacy monitoring company, very little information is given about their technology. The RDF Store is apparently named JXT, but I found no more information about it using Google.

### 3.1.8 OpenLink Virtuoso

http://esw.w3.org/topic/LargeTripleStores indicates this store handles over 1B triples. This looks like a nice commercial product. Evaluation kits are available for 15 days—not evaluated because we have no license. Supports Sparql.

### 3.1.9 AllegroGraph

Web sources and company information indicate AllegroGraph can handle billions of triples (http://esw.w3.org/topic/LargeTripleStores).

AllegroGraph Allegro graph is single threaded server based rdf store. Multi-volume support

AllegroGraph stores a triple store within a single directory (http://www.franz.com/products/allegrograph/doc/lisp/reference-guide.html).

## 3.2 Storage Engines

### 3.2.1 Full feature SQL-based Relational systems

These systems provide scaleability, multi-volumen support, transaction support, and data management tools. The systems include MySQL, Postgress, Oracle, and SQL server.

### 3.2.2 Proprietary RDF stores

Most of these systems offer little documentation that details the support given for data-management tasks, multi-volume support, and transaction support. Proprietary stores include AllegroGraph, the Sesame Native Store, and Mulgara.

# 4.0 Previous evaluations of RDF Stores

reviews several triple stores, including Jena, Kowari, 3Store, and Sesame. The triple stores were tested in their performance for three specific application tasks—'configure', 'display', and 'browse'. In all 3 tasks, when accessing a 21M triple dataset over a network connection, Sesame performed significantly better than the other contenders.

examined several RDF stores and chose Jena using Postgres for several reasons, including the existence of proven data-management tools. They found that neither Mulgara nor Sesame was as reliable and scaleable as Jena. They found that while Jena's RDF store was scaleable, its reasoner was not, and that further design decisions were needed to determine how to best support certain types of reasoning. It was also found that Joseki queries required reformulating for optimal results – logically equivelant queries could have a tremendous difference in response times. (Note that this is also true of SQL queries against an RDBMS store, though more kinks have probably been worked out over the years)

TripCom  provides a good overview of the available RDF stores and their characteristics, but does not report any peformance results.

# 5.0 Comparison Matrices

## 5.1 Storage Engine features

**Table 5.3.** Comparison of Storage Engine Features

| *Engine* | *Multi-Volume* | *Mgnmt Tools* | *Cmmty, Cmmrcl Support* | *Online Backups* | *Shadowing* | *Store Access* | *View Access* | *ACID* |
|---|---|---|---|---|---|---|---|---|
| MySQL /MyISAM | ? | Yes | Yes | Yes | Yes | Yes | ? | No |
| MySQL /InnoDB | Yes | Yes | Yes | Yes | Yes | Yes | ? | Yes |
| PostGres |  | Yes | Yes | Yes | Yes | Yes | ? | Yes |
| AllegroGraph | No | Few | Small | No(?) | No | No | No | Yes |
| Sesame | No | Some | Yes | No (?) | No | No | No | Yes (?) |
| Mulgara | No | No | Small | No(?) | No (?) | No | No | Yes (?) |
| RDF Gateway | ? | Some | Small | ? | No(?) | Yes (?) | Yes (?) | Yes (?) |
| BigOWLIM | No | Some | Yes | No (?) | No | No | No | Yes |
| OpenLink Virtuoso | ? | Yes | ? | Yes | Yes | Yes | ? | Yes |

Note that Oracle and SQL Server are not included in Table 5.3. It is assumed that, at the least, support at least the features supported by MySQL and Postgres.

## 5.2 Server/API Software features

**Table 5.4.** Server and API feature comparison

| *System* | *Creation* | *Query support* | *Transitivity* | *Reification* | *Community Support* | *Reasoning* |
|---|---|---|---|---|---|---|
| Joseki (Jena) | Yes | Sparql | No | Yes (through Jena) | Yes | OWL-DL |
| Sesame | Yes | SerQL | No | ? | Yes | RDFS |
| Mulgara | Yes | Itql | No | ? | Small | Owl-Lite |
| 3Store | ? | Sparql | ? | ? | Small | ? |
| RDF Gateway | Yes | Proprietary | RDFS Reasoning | ? | Small | RDFS (Some OWL) |
| OpenLink Virtuoso | Yes | Sparql, Proprietary | No | Yes | Commercial | RDFS |
| AllegroGraph | Yes | Sparql, Prolog | Yes (Prolog) | Yes | Small | Useful subset of OWL |

## 5.3 API/Backend Compatibility

Table does not include rows for systems that are the only users of their RDF store (i.e., AllegroGraph).

**Table 5.5.** Compatability between backends and APIs

| *System/Backend* | *MySql* | *Postgres* | *Oracle* | *Sql Server* | *Sesame* | *Mugara* |
|---|---|---|---|---|---|---|
| Joseki (Jena) | Y | Y | Y | Y | Y | Y |
| Sesame | N(1) | N | N | N | Y | N |
| Mulgara | N | N | N | N | N | Y |
| 3Store | Y | N | N | N | N | N |

Notes

1) Was compatible in version 1, but not yet in version 2

## 5.4 Query Language Comparison

**Table 5.6.** Query Language Comparison

| *Language* | *Updates?* | *Community Support?* | *Standards Compliant?* | *Transitivity?* |
|---|---|---|---|---|
| Sparql | No | Yes | Yes | No |
| SerQL (Sesame) | No | Yes | No | No |
| Itql | Yes | Small | No | No(?) |
| Prolog | No | No | No | Yes |

## 5.5 Performance Benchmarks

### 5.5.1 Data loading & Provenance insertion

Data loading and provenance insertion are evaluated by loading small RDF files, each representing a provenance record consisting of 5 triples, into the knowledge base. The amount of time it takes to load 1000 such records is compared against the current size of the knowledge base as an indication of system scalability. The results are graphed in Figure 5.1,

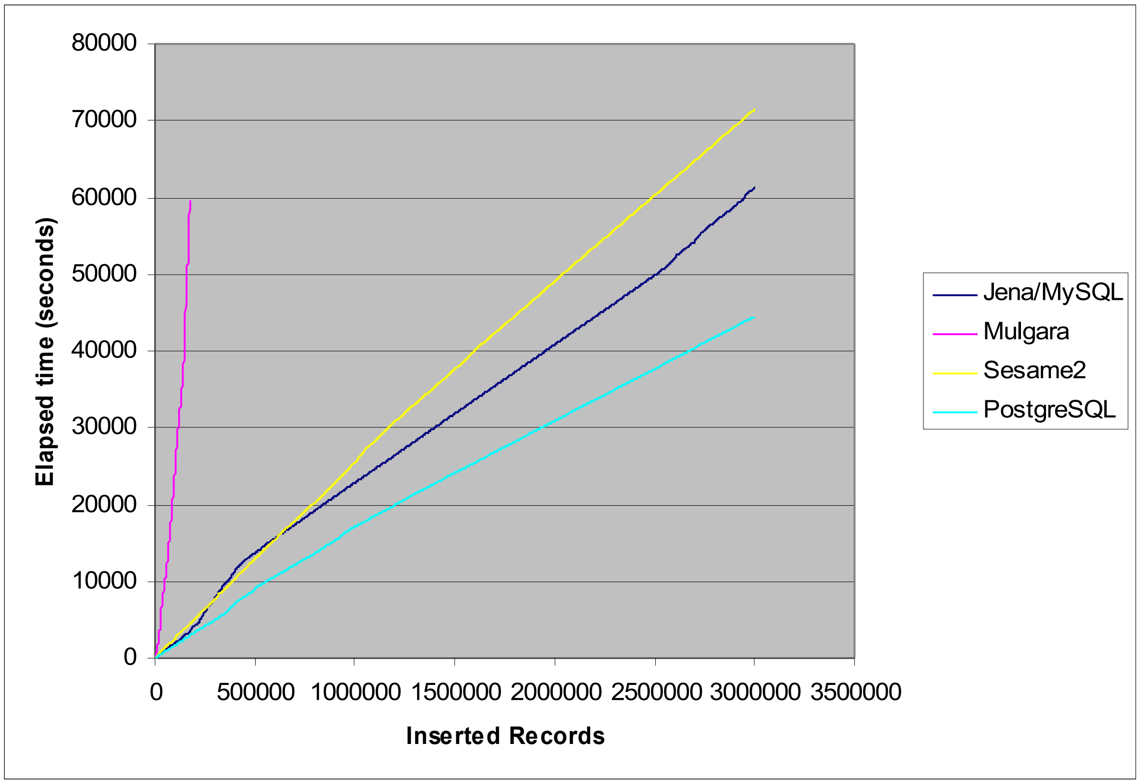

**Figure 5.1.** Data loading and Capacity

#### 5.5.1.1 Notes

Jena with PostreSQL exhibits the best performance. Jena with MySQL exhibits scaleable insertion behavior. SesameV2's behavior is also scaleable.

The Mulgara benchmark application initially aborted with an out-of-memory error after inserting 20000 records. Increasing memory for the server allowed more insertions to be made, but it still aborted after 174000 records.

AllegroGraph's documentation is very spotty on issues like backups and database parameters. I had problems setting a parameter called 'chunk size'. Setting it too small causes one kind of error, too big another kind. How to select a size is not specified, but it depends, I guess, on how many triples you plan to store. I was unable to determine a value that worked for the rdf file addition task – the server aborted if the number was too large, and created too many files if it was too small.

### 5.5.2 Loading and querying LUBM data

Different conclusions are drawn when the size of the rdf dataset is increased. Tests using the Lehigh University Benchmark (LUBM) . The LUBM

> …is developed to facilitate the evaluation of Semantic Web repositories in a standard and systematic way. The benchmark is intended to evaluate the performance of those repositories with respect to extensional queries over a large data set that commits to a single realistic ontology. It consists of a university domain ontology, customizable and repeatable synthetic data, a set of test queries, and several performance metrics..

**Table 5.7.** Load times for LUBM data

| *Dataset* | *Sesame2 Load time (Seconds)* | *Jena Load Time (seconds)* |
|---|---|---|
| 1 | 22,484 | 37,220 |
| 2 | 27,269 | 47,077 |
| 3 | 26,098 | 56,934 |

A second benchmark used LUBM datasets to compare Jena and Sesame2 in load times and query performance. Three different LUBM datasets, each with approximately 6 million triples, were loaded into Jena and Sesame2 backends. The Jena system used MySql as a backend, Sesame used it's native file store. The results, shown in Table 5.7, indicate that while Jena is slower than Sesame2, the difference is not appreciably different for the size of datasets considered. The average time for Sesame to add 6 triples to a dataset was 13 microseconds, the average time for Jena was 31 microseconds.

The query results, summarized in Table 5.8, however, indicate that there are serious problems with Jena's query engine in some cases.

**Table 5.8.** Results for queries

| | Query | Sesame 2 (ms) | Jena (ms) |
|---|---|---|---|
| 1 | ?subj <named predicate> ?obj | 235 | 395761 |
| 2 | <named subject> $pred $obj | 204 | 812 |
| 3 | $sub $pred <named object> | 188 | 860 |
| 4 | $sub <named predicate> <named object>. $sub $pred $obj | 203 | 750 |
| 5 | $sub $pred $obj FILTER($obj='Literal') | 187 | *error: Out-of-memory* |
| 6 | $sub $pred 'Literal' | 188 | 593 *error: no results* |
| 7 | $sub $pred $obj FILTER regex($obj, 'Literal.') | 187 | *error: Out-of-memory* |

# 6.0 Conclusions

## 6.1 Backend Selection

In the near future, we are still working with prototypes and data, and data integrity is not a serious issue. The large scale LUBM benchmarks show that the Sesame2 native store's performance is orders of magnitude better than the current database backends in query performance, so it will be used. Perhaps Sesame2 will support a different backend by the time we need it.

In the long term, a backend that uses a standard industry database, such as MySQL, Postgres, or Oracle is desired. Systems using native backends do not have the history that gives our team confidence in they're ability to provide database management tools, access control, 24X7 access, online backups, etc. Jena has recently provided an additional backend which can use commercial backends and is optimized for use on SPARQL Queries which may fit the bill . In the long term, using Oracle as the backend is desired, since it is forseen that many customers will have experience with supporting Oracle. MySQL will be considered because its open source, it is installed, and the team is familiar with it.

## 6.2 API Selection

Two APIs have strong community support and meet the requirements of the team: Jena and Sesame1. Both can use MySql as a backend and both have similar strengths in supporting queries and in manipulating RDF graphs. The other APIs seem are either only available commercially, have limited community support, or are tied to proprietary backends.

Jena's strengths are its support for a wide variety of backends, strong community support, and support for complete OWL-DL reasoning. Drawbacks include a perception of over-complexity of the API and weaknesses in the query optimizer (logically equivalent queries can result in different execution times).

Sesame’s strengths include strong community support, reported faster access speed, and previous usage at PNL. Its main drawback is lack of support for an RDBMS backend for the current release—this makes direct performance comparisons difficult.

Given these difficulties and the functional similarity between the two APIs, both APIs will be supported for the nonce.

## 7.0 References