SCAM Portfolio Scalability

Henrik Eriksson
Per-Olof Andersson

Uppsala Learning Lab

2005-04-18

Contents

1 Abstract

The SCAM Portfolio is a flexible portfolio system based on RDF. In this report the performance and general scalability of the portfolio system have been analyzed.

The current performance is far below what's needed for large scale use, such as integration with the Uppsala University portal. This concerns both searching and browsing the portfolio.

Several measures are suggested to improve the performance, maintainability, and the ability to integrate with the University portal and file server.

The suggested performance improvement for searching consists of taking advantage of the internal search engine's ability to delegate searching to several dedicated search processors. This would have several advantages and it's likely that a satisfying performance for searches could be achieved.

The suggested performance improvements for browsing include caching of components, changing the database layout, and enable distribution of the database. Although these improvements are promising, it's difficult to say whether they will be enough to reach a satisfactory performance.

2 Suggested Improvements Summary

In order to improve the performance and general scalability of the SCAM portfolio system, the following measures are suggested:

Measure	Estimated time (months)
Change database layout to incorporate component membership, to create more efficient database access. Improve the translation of RDQL queries.	3
Introduce caching of components to reduce the number of database requests.	0.5
Improve the tools available for administration of the portfolios.	1
Enforce unique identifiers for portfolios and users to improve maintainability and portability of portfolios.	1
Enable the database to be distributed in order to improve reliability and performance.	0.5
	= 6

In order to integrate the portfolio system with the university portal and the existing file server, the following measures are suggested:

Measure	Estimated time (months)
Integrate the user authorization of the SCAM portfolio with that of the university portal.	1
Avoid streaming of files through the portfolio middleware in order to be able to efficiently redirect file uploads/downloads to a dedicated file server.	2
Redirect files uploaded to the portfolio to the university file server. Enable public sharing of uploaded files. Integrate access to the file server in the portfolio interface.	2
Resolve access rights in the university file server so that files that are made public in the portfolio interface become accessible from the file server.	IT-stöd?
	= 5 + IT-stöd

3 Abbreviations

The following abbreviations are used in this document:

- **API** Application Programming Interface
- **IEEE** Institute of Electrical and Electronics Engineers, Inc.
- **J2EE** Java 2 Platform, Enterprise Edition
- **JAAS** Java Authentication and Authorization Service
- **FOAF** Friend Of A Friend
- **LOM** Learning Object Metadata
- **OSPI** Open Source Portfolio Initiative
- **RDF** Resource Description Framework
- **RDQL** RDF Data Query Language
- **SCAM** Standardized Content Archive Management
- **SHAME** Standardized Hyper Adaptable Metadata Editor
- **SQL** Structured Query Language
- **URI** Universal Resource Identifier
- **W3C** World Wide Web Consortium

4 The SCAM Portfolio System

4.1 Architecture

The portfolio system is a J2EE application based on SCAM, an RDF-based metadata management system:

`http://scam.sourceforge.net/`

Editing in the SCAM portfolio system is performed with an integrated version of the RDF editor SHAME:

`http://kmr.nada.kth.se/shame/`

SCAM is based on RDF, which is a format for flexible description of *metadata*, i.e. data about data. RDF is developed by the same people that developed HTML, and is described here:

`http://www.w3.org/RDF/`

A *very* simplistic description of RDF is that RDF provides a flexible way of describing properties. RDF consists of triples where the first part specifies what or who has the property, the second part specifies the name of the property, and the third part specifies the property value. RDF is developed as a part of a bigger project, named the "Semantic Web":

`http://www.w3.org/2001/sw/`

4.2 Features And Comparison With Other Portfolio Systems

The SCAM Portfolio provides basic portfolio features, such as the ability to create folders and subfolders, upload files, and creating links. All portfolio items are typed and can be annotated with type-specific metadata.

There are other open source portfolio systems available with comparable features, e.g. OSPI:

`http://www.theospi.org/`

The main thing that separates the SCAM portfolio from other portfolios that are not based on RDF, such as e.g. OSPI, is mainly flexibility. The SCAM portfolio uses this flexibility to enable a number of things:

- Extensibility
- Powerful type hierarchy
- Resource relations
- Standard compliance

4.2.1 Extensibility

The extensibility comes from the nature of RDF. RDF is focused on individual properties and introducing a new property doesn't break the existing metadata design. In addition, new properties can be related to existing properties with sub-property-of relationships, which provides old viewers and editors with a fallback strategy for handling new types of data that they don't fully support.

4.2.2 Type Hierarchy

All resources, i.e. items and files, in the SCAM portfolio are typed. In RDF type membership simply means adding a property that a resource has a certain type. A resource can have more than one type and types can be subclasses of more general types.

The type of a resource is used to determine what kind of viewer to use for displaying metadata about the resource and what kind of editor to use to edit the data. The editors and viewers in the SCAM portfolio are modular and are dynamically assembled on the fly, based on the types of the resource. As an example, a resource typed as "text" might have the properties "author" and "title". The type "book" might extend the "text"-type and include an ISBN number. Whenever a resource typed "book" is edited an editing form is created based on the properties associated with the type "book" and all of its superclasses, in this case the "text"-type.

It's also possible to reuse parts of editing forms. If two different types need a specific property, they can both link to that editing sub-form.

Which properties that are actually associated with each type and how types are related to each other is managed by a configuration file that is easily extended and modified.

An important thing to note here is that viewing or editing a resource as a certain type doesn't delete any metadata that is not implied by that type. The editors of the SCAM portfolio leave all metadata they don't understand untouched.

4.2.3 Resource Relations

In the SCAM portfolio it's possible to introduce relations between resources. Examples include links and annotations, but the flexibility of the SCAM metadata makes it possible to create new relations as the need arises.

4.2.4 Standard Compliance

An important aspect of RDF is that properties and types have real URI:s, which means that they are globally unique. Simple strings like "creator" doesn't provide clear semantics. Is the creator of a book the author or the publisher e.g.? By having properties with URI:s and by providing a clear definition of the properties any ambiguities can be resolved.

Of course it's easy to invent your own properties and types but whenever possible it's best to use the available standard ones. It is easy to incorporate standard ontologies into SCAM, and the most common ones are already available, such as LOM, Dublin Core, vCard, and FOAF.

Unique URI:s for types and properties together with standard ontologies are important factors when relating portfolio data to the outside world. As an example, it's possible to connect a portfolio system to other networks such as e.g. the Edutella network, which is an international peer-to-peer network for searching semantic web metadata. The use of standardized properties is a necessity to be able to understand metadata across systems.

5 Performance

5.1 Current Performance

The current performance of the SCAM Portfolio was measured by using the Java based load-testing framework Grinder:

`http://grinder.sourceforge.net/`

In both test cases the portfolio system was installed on JBoss version 3.2.1 deployed on Tomcat version 4.1.24.

5.1.1 Test Case 1

The first test case was a small portfolio installation with approximately 300 portfolios. The in-memory database, Hypersonic, run on the same server, was used for data storage.

The tested portfolio was installed on a 1.8GHz, 2GB RAM AMD Athlon MP, with two processors, running a 2.4.26 linux kernel.

Four types of test were run; two searches and two browsings. The searches consisted of finding all occurrences of the terms "KMR" and "math", respectively, which in the tested portfolio would produce 38 and 144 hits, five of which would be retrieved and shown.

The browse tests consisted of opening the top-levels of two user portfolios which contained a root item together with 1 and 8 immediate child items, respectively.

Each test was performed with both 10 and 100 simultaneous requests, and all tests were run 10 times consecutively.

	No. of parallel requests	Mean time to first byte (seconds)
Search 38	10	0.3
	100	4
Search 144	10	0.3
	100	5
Browse 1	10	0.4
	100	3
Browse 8	10	0.4
	100	4

Table 1: Test times for searching and browsing a SCAM portfolio system.

As *Table 1* shows, the times for search and browse were approximately the same. Since a search consists of retrieving the five topmost search results in addition to performing the search it's almost a superset of the browse case. The numbers indicates that retrieving components is the most intensive part of the work.

5.1.2 Test Case 2

The second test case was a simulation of a full university installation with 30000 portfolios (the approximate number of Uppsala university students). Each portfolio had 10 text items and 10 folders at the root level. Each folder contained 20 text items. Each folder and text item had 1-3 random words picked from an English dictionary as title, and 1-10 random words

as description. A PostgreSQL version 7.4 database run on the same server was used for data storage. The database required approximately 30 GB hard disc space, which means that each portfolio required about 1 MB storage.

The server machine was a 2GHz, 1GB RAM AMD K7, running NetBSD with kernel linux emulation.

Two types of tests were run, browse and search. The browse test opened a random portfolio and retrieved the top level 10 items and 10 folders. The search test searched for all items from a random portfolio, resulting in 210 hits.

Each test was performed with different number of simultaneous requests, each run 20 times consecutively.

	No. of parallel requests	Mean time to first byte (seconds)
Search	1	0.96
	5	6.5
	10	29
Browse	1	0.85
	5	3.9
	10	7.7
	20	15
	30	24

Table 2: Test times for searching and browsing a SCAM portfolio system.

As *Table 2* shows, the times for search and browse were significantly worse than in test case 1. The biggest reason for the increased time of browsing compared with test case 1 is the time to retrieve items in a big database stored on the hard drive compared to the small database stored in RAM memory in test case 1.

The reasons for reduced search performance compared with test case 1 is the big difference in search material. The search engine in the portfolio system is Lucene, `http://lucene.apache.org/`, and it works by creating an index over all searchable material. In the portfolios all items are regarded as documents searchable by Lucene, which means that there were over 6 million documents with all their text strings added to the index. The search index for this portfolio installation was 4.1 GB, and since that is too big for keeping in RAM memory, all searches had to be performed by querying the index on the hard drive.

5.2 Performance Issues

The SCAM Portfolio system is based on RDF, which is stored in an underlying relational database. The database layout is essentially a single table with three columns for each part of the RDF triples. A fourth column is added to provide information about which user portfolio an RDF triple belongs to.

Although the SCAM portfolio system is RDF based, RDF triples are not retrieved individually from the database but generally in groups corresponding to a *component*. This is a SCAM specific term which roughly translates into "a set of RDF triples belonging to one portfolio item". Portfolio items are the folders, files etc. that are visible in the portfolio view.

When a request needs an item to be retrieved from the database, the RDF triples corresponding to that item must be fetched in several steps, each step producing some triples

which in turn indicates which additional triples need to be fetched.

This means that several requests are made to the database for each item to be retrieved, which could be a source of performance loss.

Another potential factor in the scalability of the portfolio is the upload and download of files to the system. All files are currently streamed through the portfolio server, which could be a problem if one would like to decouple the file storage from the portfolio server to a separate file server.

As a final note there is an issue in SCAM with the RDQL translation into SQL. RDQL is a query language used for asking complex RDF queries. It's not used at all in the current stable version of SCAM Portfolio, but it's available as a service for optional external use. When tested with a development version of SCAM which uses it internally to find annotations it was shown to be extremely inefficient due to unnecessary complex SQL translation.

5.3 Suggested Improvements

To improve the performance of the portfolio system the following changes are suggested:

- Add a column to the database RDF table indicating which component the RDF triple belongs to.
- Introduce caching of retrieved components.
- Eliminate file streaming through the portfolio server in order to be able to separate file handling to a dedicated file server.
- Improved RDQL-to-SQL translation.

Adding a component column to the database RDF table would make it possible to retrieve a portfolio item (component) with one database request, which would increase performance. This would imply changing some portfolio API:s to take this into account.

Caching of retrieved components could improve performance by not having to search the database for frequent requested components.

Eliminating file streaming through the portfolio server would be necessary if one would like to manage uploaded files on a dedicated server, since the benefit would otherwise be lost by having the portfolio server duplicating all streaming work done by the file server.

The RDQL-to-SQL translation needs to be improved. The currently used multiple-joins of the main SQL table for even the simplest RDQL queries needs to be translated into something more efficient.

5.4 Performance Correlation With Number Of Users

It's difficult to translate the results of the test cases into a meaningful measure of the amount of users the portfolio can support. In order to do that one has to have information about the users: How many of the potential users are going to actually use the portfolios? How many users are going to be logged in at the same time? How many of the logged in users are actively clicking on things at any moment and what is the highest average clicking frequency of these users? Unfortunately these figures can only be guessed.

The performance measurements show that the portfolio can process approximately one request per second for both searching and browsing.

If one would take the waiting time of five seconds as the maximum tolerable waiting time, and one assumes that 75% of the potential users use the portfolios and that at most 50% of them are logged in at the same time and that at most 1% of those users are simultaneously clicking on things, and that the highest average click per active user is one click per five seconds, then the tested portfolio would support:

$$\frac{5}{0.75 * 0.5 * 0.01} \approx 1300 \text{ total users}$$

which of course is far less than the potential 30000 users of the University students. But again, this is based on very uncertain estimates.

5.5 Will The Suggested Improvements Be Enough?

The test cases shows that the current performance is far from what is needed for both search and browsing.

For the search case there is reason to be optimistic. The internal search engine, Lucene, supports splitting up its search index and using multiple processors to perform the search in parallel, which will give several advantages. Executing the search on multiple processors in parallel means that the search time is divided with the number of processors. Furthermore, these processors will take the work load off the portfolio processor. Another benefit is that the search index can be divided into small enough parts to fit inside each search processors (RAM) memory, which is much faster than accessing the index on the hard drive. The search index for the 30000 portfolios is 4 GB which means that using 2-4 dedicated search processors would be enough for the index to be stored in memory.

For the browse case it's a lot more complex. There are several suggestions for performance improvement in this report, but it's difficult to say how big their impact on the performance will be before they are implemented.

The effect of caching components rely heavily on how the users use the portfolio, since the benefits of caching relies on the same components being requested frequently.

Adding a component column to the database would both reduce the number of SQL queries needed to access a component in the database, but also enable clustering optimization of the database so that database rows belonging to each component are juxtaposed for improved access speed. But it's difficult to give an quantification of the improvement this would mean.

6 Administration

Scalability of the portfolio system is not limited to performance alone. A large user base means a lot of administration. At present administration is done manually through a web interface, which is adequate for a small amount of users, but probably not practical for administrators of a large system. What's needed is a tool for automatic administration, such as e.g. bulk registration of a group of users etc.

Another issue is the names of different portfolios. At present users and portfolios are identified by a string chosen by the administrator. The identifiers of all items created by that user is based on that initial string which means that if a user is removed from the portfolio system and another user is added with the same name, all references to items created by the first user may now point to items created by the second user. This is clearly unwanted, and

in addition if a user exports her portfolio and wants to import it into another installation of the portfolio system there may already be a user with that identifier.

In order to avoid such clashes portfolios and user identifiers should be made globally unique. To support and enforce this the administration tools of the portfolio need to be modified.

7 Reliability

The current portfolio system uses a single database for storing data. If the server malfunctions it will take down the whole portfolio system. Unless the database becomes corrupted for some reason, restoring the system will consist of simply restarting the server. If backups of the database are made each night even a corrupted database will be able to be restored, but would of course lose all changes made since the last backup.

A more reliable option would be to distribute the database over several servers so that even if one server goes down others can pick up and restore the data. One possible change to the current portfolio would be to use several database servers, and whenever data needs to be written it's written to all of them, but when the data needs to be read it's only read from one of them. This would improve performance and reliability.

8 Integration

8.1 The University File Server

The existing university file server is maintained by IT-stöd. Access to files on that server by users is done either through a web interface or via Samba.

At present the SCAM portfolio system maintains files uploaded to it on its own server. In contrast to the existing file server, files uploaded to the portfolio system can be made publicly available at the user's discretion.

One possible solution to integrate these two file servers would be to keep the existing file server and to redirect files uploaded to the portfolio to be stored on the file server instead of the portfolio server. A problem with this approach is that the portfolio system keeps metadata about files uploaded to it, and therefore needs to be able to find the original file. If someone accesses the file server via Samba and moves the file or renames it, the portfolio system needs to be able to find the file.

One possible solution to this is to use the fact that the existing file server is running a Unix-like operating system where files can be referenced by a Inode identifier. This remains the same even if someone moves or renames the file.

It's not fool proof though. If a user deletes a file and recreates another file with the same name, it will get a different Inode identifier and the portfolio system will not be able to find the original file.

Apart from changing the internal file handling in the portfolio it would be useful to create a unified web interface for the file server and the portfolio.

8.2 Common Log-In

The portfolio system has its own authentication and log-in for users. If it is going to be included in the university portal it would be practical for users to only have to authenticate

themselves once.

At present the portfolio system is using JAAS authentication and with a little work it will be possible to integrate that with the authentication system of the university portal.

8.3 Coordination With IT-stöd

This is a summary of IT-stöds view of a portfolio system in the new university portal:

IT-stöd will help out with two tasks if the new Student portal is to contain a portfolio system:

- The scalability and capacity (amount of concurrent users) of the system must be known and meet certain criteria. IT-stöd can get statistics out of the present system to get a rough view of how many users should be calculated with. IT-stöd can probably help out with load tests of the portfolio system if necessary.
- IT-stöd today runs a service called "personal file area" which gives each student access to 150 MB of disk space. This diskspace can be accessed through https, ftp and samba (Windows file sharing). The file area of today is purely personal and offers no possibilities for sharing or publishing files.

A portfolio system has some similarity with this "file area" but offers much more functionality. The introduction of a portfolio system for the students would therefore make the present file area obsolete from the portal view, but a portfolio system can not easily be used in Windows file sharing. The portfolio can contain many types of objects, of which only objects of type "file" should be published in the Windows file system view. The portfolio can also keep metadata about each file. The transfer of a file from the portfolio to a Windows file system would drop this metadata. There are also purely technical concerns that has to be solved if a portfolio system and the Windows file sharing should be integrated; the portfolio does not store files in a file system that reflects the topology presented to the user. The hierarchy of folders is kept in a database which keeps pointers to files stored in a flat file structure. This makes it impossible to use the Samba system out of the box.

IT-stöd will help out with the decision on what to do with the present "file area". If it is to be integrated with the portfolio system, some development work must probably be done by IT-stöd.