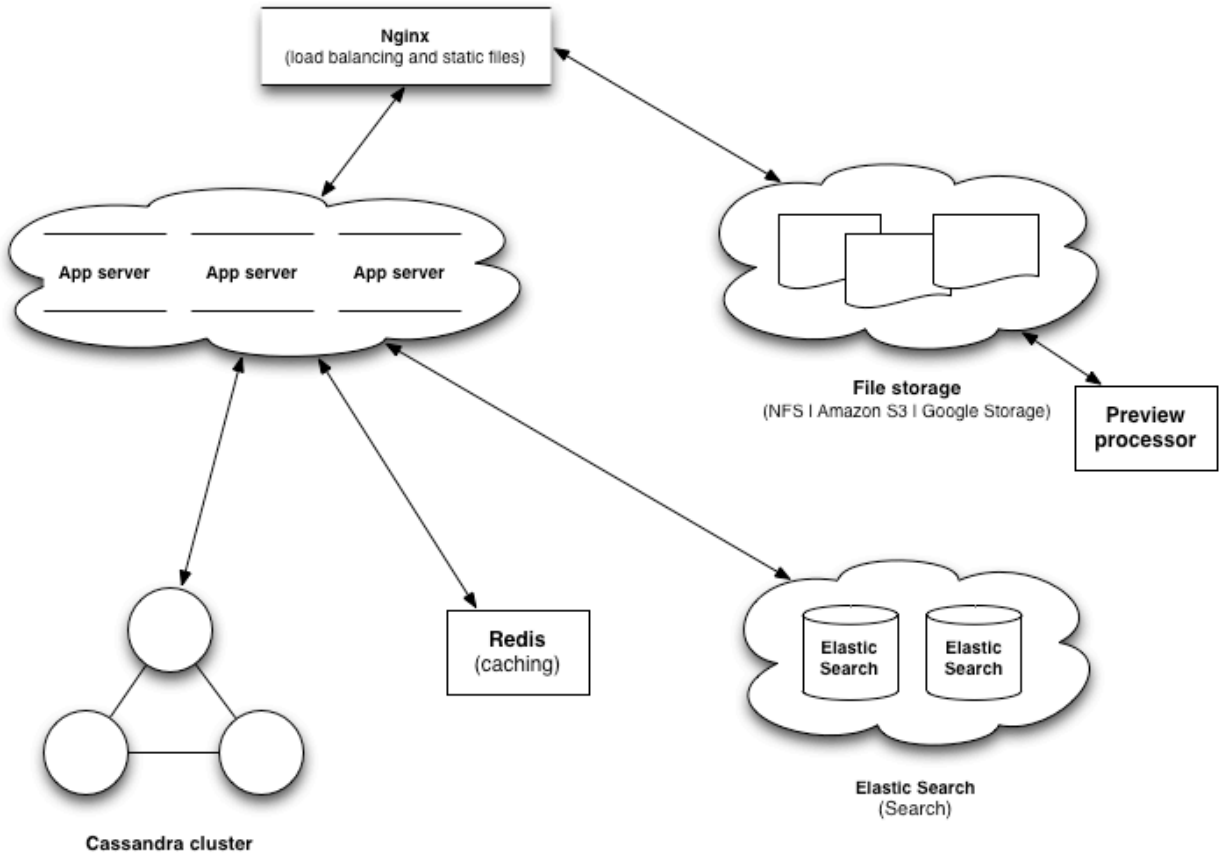
Sakai OAE re-architecture

Progress update
November 15, 2012

1. Architecture update and technology choices

The technology stack as outlined in our previous progress report remains largely unchanged. However, two modifications are worth noting:

- **Search:** Apache Solr has been switched out in favor of ElasticSearch. The main reason for making this change is the fact that ElasticSearch has been designed to be cloud aware, has built-in AWS EC2 knowledge and rack awareness, shards and partitions indexes out of the box, and provides horizontal and elastic scaling. This is all possible with Solr as well, but requires quite a bit more configuration to get things right. ElasticSearch also seems to be dealing better with a large number of query terms, and has some interesting multi-tenancy capabilities that we might want to use later on.
- **File storage:** Next to NFS and Amazon S3 storage, which were the storage solutions listed in the previous architectural diagram, we have also made it a goal to also natively support Google Cloud Storage, and provide extension points that make it easy to integrate with other content storage providers.



2. Progress

Progress has been made in a number of areas which we outline below. A lot of the work has focussed on foundational components, and quite a few of these are currently under review. Most of them have also had initial performance testing. The code should be merged into the codebase fairly soon and will then undergo further performance testing and optimizations where necessary.

2.1. Performance testing and results

A reference deployment of the new technology stack has been set up on the Joyent Cloud and is currently mostly used for ad-hoc and nightly performance testing. All of the individual components have been puppetized so they can easily be replicated, and the scripts for this can be found in the https://github.com/sakaiproject/puppet-hilary repository. Most of the components are being used out-of-the-box and have not been tuned. The reference deployment is using the following machines:

Type	Spec	Cost
Front-end	1 XS (0.5 GB / 1 CPU) instance	$21.90 / month
App servers	2 Medium (1 GB / 2 CPU) instances	$124.10 / month / inst
Database	3 Large (8 GB / 2 CPU) instances	$262.80 / month / inst
Caching	1 XS (0.5 GB / 1 CPU) instance	$21.90 / month
Search	1 XS (0.5 GB / 1 CPU) instance	$21.90 / month
		Total: $1,102.30 / month (excluding content storage)

The performance testing framework that has been set up consists of 2 parts, a data load and a Tsung performance test. First, a set of snapshotted data is restored when the environment is redeployed. This currently consists of 40,000 users, 80,000 groups and 200,000 content items. Once this is done, the OAE Model Loader (https://github.com/sakaiproject/OAE-model-loader) will generate a new set of 10,000 users, 20,000 groups and 50,000 content items and will load these into the system. This data load is monitored and allows us to check the speed at which data can be inserted into the system.

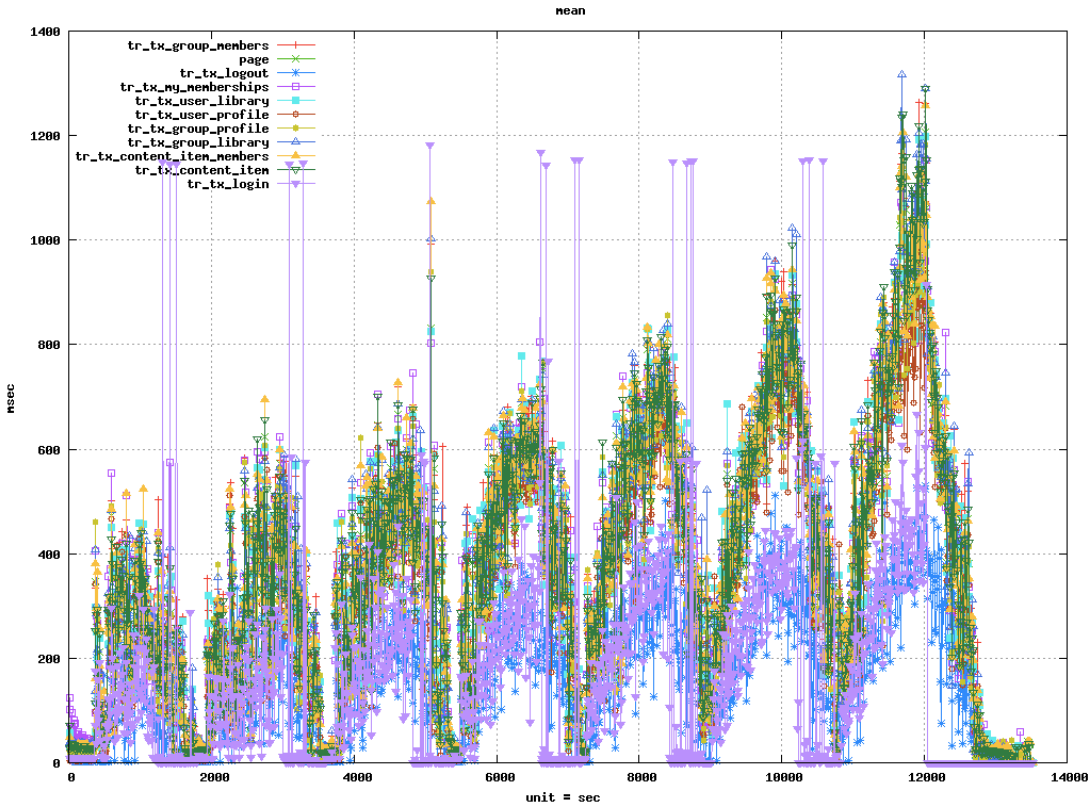
In order to do the actual load testing, we have decided to continue using Tsung, which is an Erlang-based load testing tool that simulates user sessions and provides various reporting capabilities. An additional Node.js module has been written that facilitates the creation of Tsung XML test files, and is used to create a large number of random user sessions that reflect real user behavior as closely as possible. Unfortunately, all of this is based on best guestimates as we don't have access to any OAE production logs.

Once all of the data is loaded into the system, the Tsung load test is kicked off. Various waves will be run during which additional sets of users are added until we find the breaking point (for the current infrastructure). The reporting is done twofold, by using the Native Tsung reports and by pushing Telemetry data into Circonus (see below). A separate blog post will be written that goes into more detail on all of this.

When the first tests were run, we were able to quickly identify a number of performance issues in permission checks. Various solutions have been tried and we believe that these issues are now fixed.

This has allowed us to set a performance baseline that we'll be validating against as we move forwards. The Tsung results for this baseline test can be found at http://165.225.132.68/2012/11/13/17/00/tsung/20121113-1730/report.html and the Circonus average response times chart for this test can be found at https://circonus.com/embedded/graphs/5bbd2647-be58-6419-eeea-90ee7bb1cef5/68D29H.

The test is comprised of a number of waves that have an increasing number of new users arriving every second. The first wave has an arrival rate of 4.6 new users/second, which is incrementally increased until it hits the last wave at 13.6 new users/second. As shown in the chart below, that final wave is the first one that crosses the 1 second mean transaction (page load) time threshold, even though it's not making the servers unstable yet.

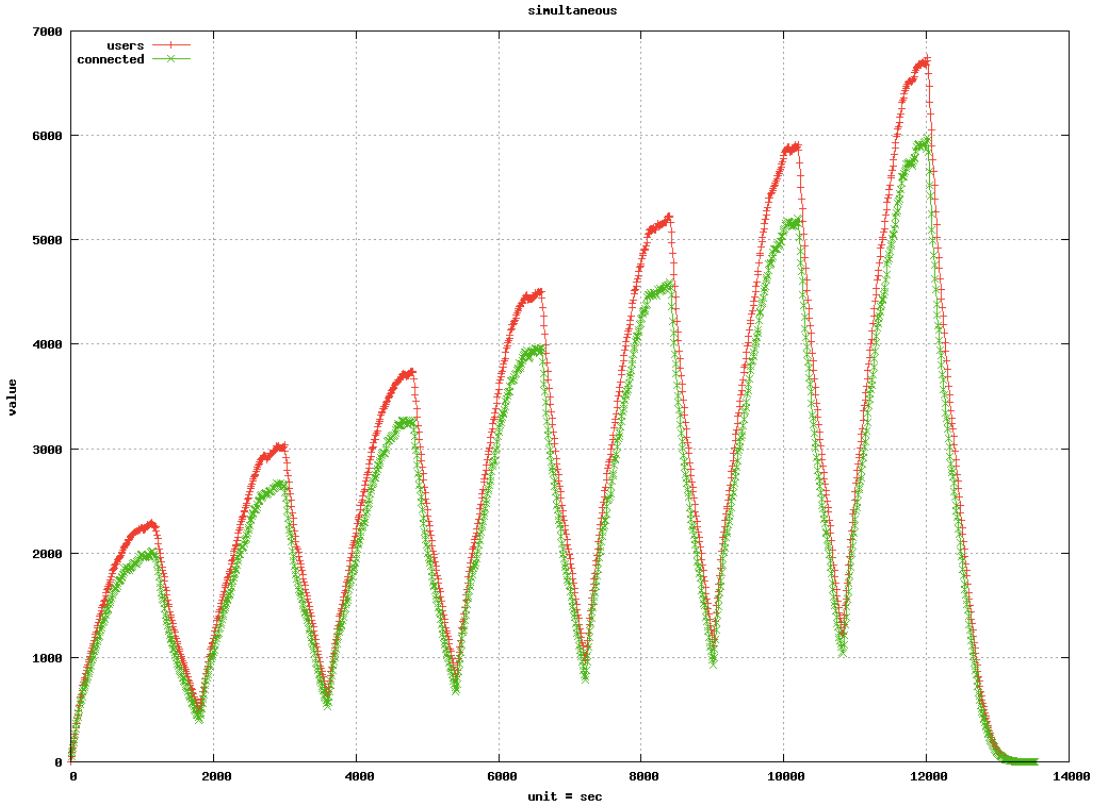


This generates the following performance baseline for the infrastructure that's being used:

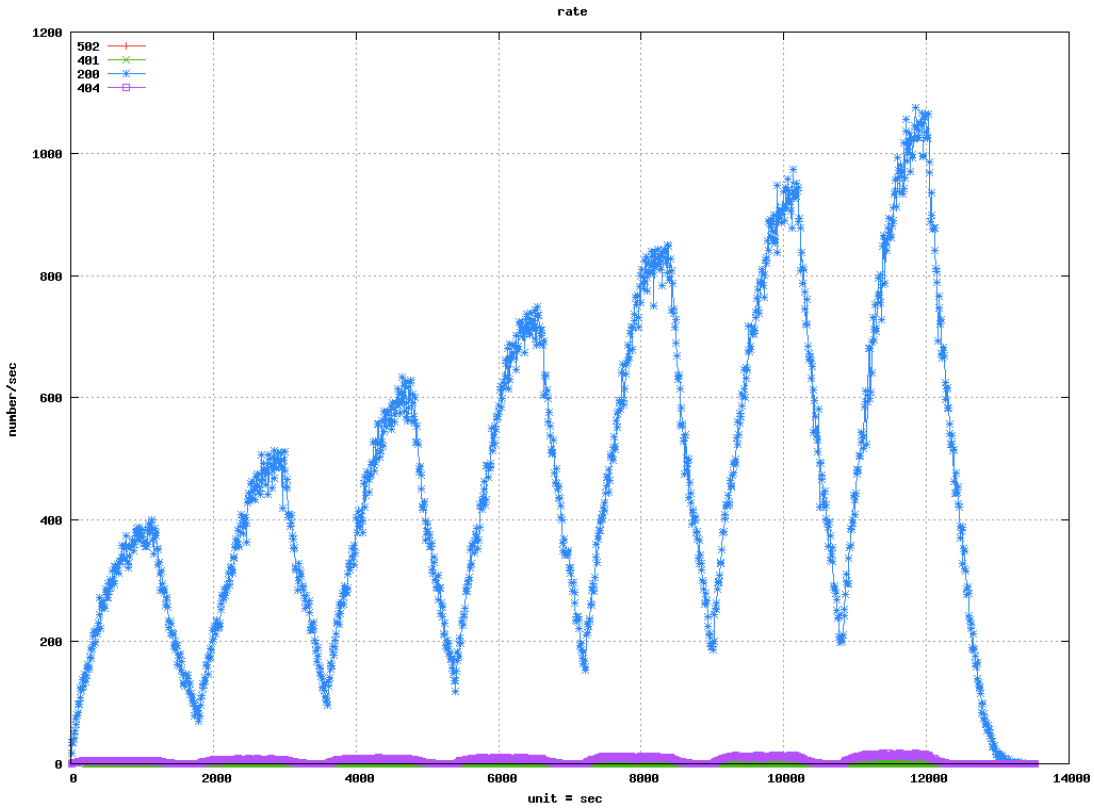
Metric	Result
Simultaneous users	+- 5,800 users
Requests per second	+- 1,050 req/s

An hour long performance test with an arrival rate of around 10 new users/second manages to remain stable throughout and ends up doing a little under 4 million HTTP requests during that time.

Obviously, these tests are only testing the functionality that's already been re-implemented. Once new features are implemented, the baseline will either go up or down, although efforts will be undertaken to keep the numbers as close as possible to what they are currently.



Concurrent users

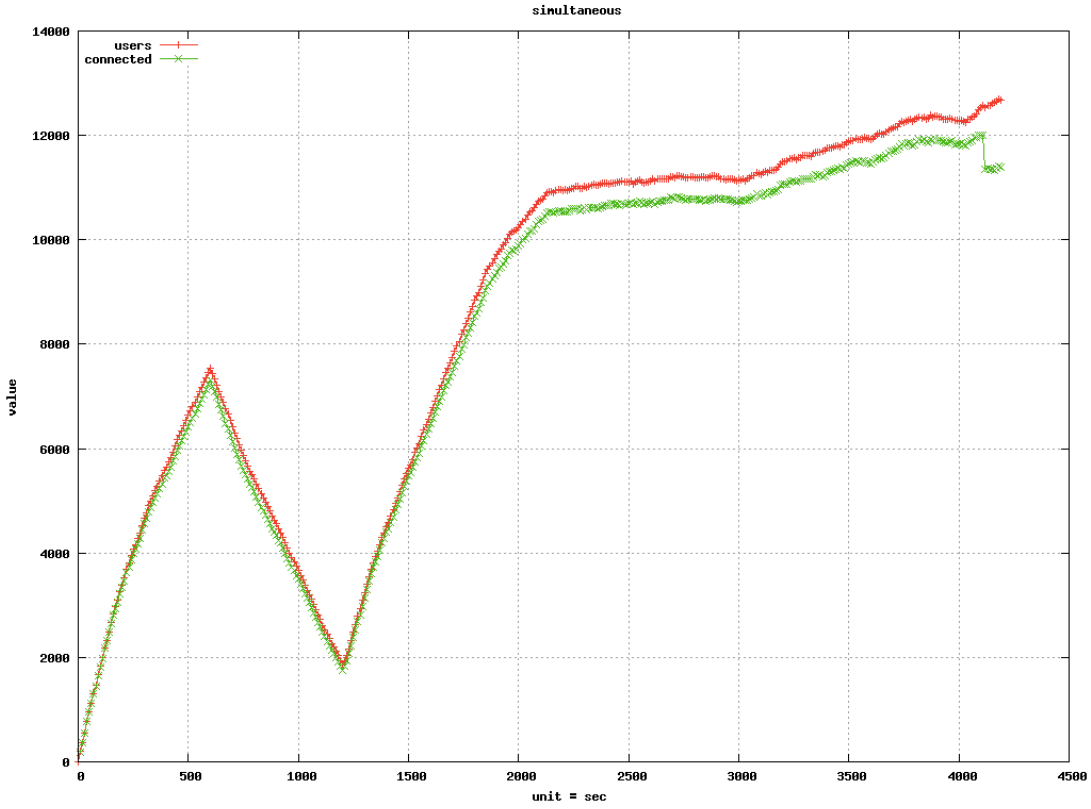


HTTP requests per second

In order to investigate the linear scalability characteristics of the current architecture, we have doubled the number of app servers and have added 2 additional Cassandra nodes. This generated the following results:

Metric	Result
Simultaneous users	+- 12,000 users
Requests per second	+- 2,000 req/s

This shows that doubling the available infrastructure will double the system's capabilities and we can thus be fairly confident that the system will scale horizontally in a linear way, which is in line with expectations as the technology choices have been made with this in mind. Obviously, in the next month we will be performing these tests with search and file uploads included as well.



2.2. Search

Search has been implemented into the Hilary back-end using ElasticSearch, and is currently under review. Like Solr, ElasticSearch is a Lucene-based free-text search server that can be accessed via a RESTful API. ElasticSearch has been built from the ground up with the elastic scalability expected from a cloud deployment in mind, and provides a variety of great tools that make administration of such a cluster of nodes easier. There are additional features such as dynamic index creation, schema-less design, and cross-index searches that make it an extensible and flexible choice.

The OAE "General Search" has been implemented in the Hilary backend, and there are more planned in the design (Library search, Memberships search). The General Search implementation exercises some of the higher risk aspects of search such as access-scoping searching, and the stability and performance of the search platform itself. After a series of performance tests, we've found that ElasticSearch's performance itself hasn't proven to be a bottleneck to the system, however more testing of its limitations is still expected.

2.3. File storage and preview generation

There is currently a pull request under review for the re-introduction of file uploads, including file revisions, using Node.js Formidable to handle the file uploads and Nginx to handle the downloads. Two initial implementations have been created, one that allows local NFS storage and one that allows storing data into Amazon S3. Later on, Google Cloud Storage will also be provided.

Preview and thumbnail generation has also been re-implemented, based on the rSmart/Erik Froese Nakamura preview processor. It uses RabbitMQ to queue the content items that need preview processing and can thus be horizontally scaled.

This component has been built with extensibility in mind, and provides an interface that allows for new storage providers to be integrated.

2.4. UI integration

An initial set of UI integration work is currently under review as well. It re-enables basic functionality like creating an account, logging in, logging out, profiles, my library, my memberships, creating a content item, managing a content item, etc. This will be added to the reference environment in the next week or so.

We have also started refactoring, cleaning up and simplifying all of the client-side Sakai APIs, in order to reflect the refactoring of business logic previously embedded in the UI but now removed to the back-end. This should also help make widget development more straightforward.

Widget development will also start using the node package manager (NPM) for distributing and installing widgets.

The existing UI production build system, which concatenates, minifies and hashes the UI files has been re-implemented using Grunt.

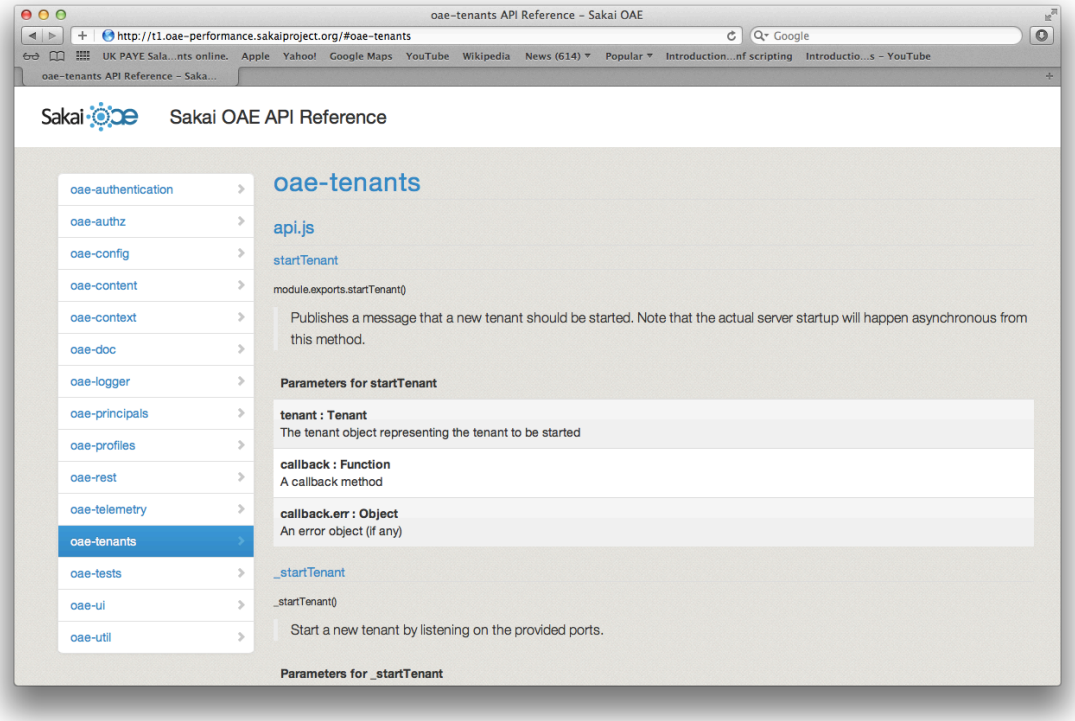
2.5. Admin UI

The administrative UI has been continuously extended to include more configuration types (array configurations, tree configurations, etc.). A lot of the existing configuration options that were part of the client-side configuration file have been moved into back-end config descriptor files, which will allow for the retirement of the client-side config file.

2.6. Documentation UI

Using the Node.js Dox module, a User Interface has been created that ships with the main codebase and shows the documentation of the Hilary API functions. This should make it easier for (widget) developers to discover and use the available API functions. The current documentation UI can be found at http://t1.oae-performance.sakaiproject.org/.

There is currently also a pull request under review that significantly improves the quality of this documentation.



2.7. User ID association

The Hilary back-end has moved away from using the login id as a user's internal id, and a pull request is currently under review where all users have an internal id that can be mapped against one or more login ids. These login ids can be institutional ids (SSO), Google, Facebook, Twitter, etc., depending on which authentication providers the tenant allows.

This makes it for example possible for a user to log into the same account using his institutional id and his Google account, or for a user to continue using the same account after his institutional id has changed.

2.8. Unit testing

The unit testing suite has been re-factored to use the REST endpoints where possible. This has led to the creation of an OAE REST module, which describes all of the available REST endpoints and makes it easy for deployers to create, add, update or remove users, group, content, permissions, etc.. This module can be pulled out into other projects and can be used to easily interface with a running OAE instance.

There's been a continuous effort to increase the unit test coverage, which currently stands at 92%.

2.9. User and group provider design

Conversations have started about the need for a User provider and a Group provider for Sakai OAE. There was an initial conversation with people from Edia about this, where they shared their previous experiences integrating Sakai CLE and Sakai OAE with OpenConext. We agreed to continue working together on a design for these providers, and to publish these to the wider community after the first iteration. The idea is to take the Sakai CLE User and Group providers as a starting point, as these have been production hardened and have been valued because of their extensibility.

Edia has shown interest in validating the design and implementation by creating an OpenConext integration.

3. Next steps

Between now and Christmas, the team will be focussing on the following tasks:

- Continued UI integration work and client-side API improvements. Once search and file upload are merged, the UI should be hooked up to these as well.
- Improved tenant separation by allowing tenants to be completely private to other tenants.
- Implement additional search endpoints for Library search and Membership searches.
- Continued performance testing for file storage, preview generation and search.
- Implementation of commenting capabilities, including replying to comments.
- Implementation of activities, based on http://activitystrea.ms/specs/json/1.0/.
- Design of Sakai Doc REST API.

This is our current best estimate of where we think we can be by the end of the year. This would be quite significantly ahead of the plan made in the beginning of September and would represents a large part of the existing OAE functionality. However, it is obviously not yet feature complete.

We still recommend the end of the year for a new go-ahead decision based on the progress made against the points mentioned above. If there is a decision to move ahead, a prioritization exercise should be organized to determine which new features and capabilities should be implemented first.