Sakai 2.5 Performance Testing at Wiley

Recently, Unicon, Inc. and John Wiley and Sons, Inc. undertook a serious round of performance testing of a revision of the 2.5.x branch of Sakai. We have other rounds of performance testing planned, and the purpose for this first round was to obtain baseline performance data before we make changes to the application. Below you will find a synopsis of the testing, and you'll also find the complete documents for the test plan and the test results report. If you have comments, questions, suggestions, or criticisms please share them with us via the production mailing list.

Test Plan

We had two objectives for this first round of testing. First we wanted to validate that the core Sakai framework and a particular set of tools met minimum performance criteria. Second, we wanted to establish a baseline set of performance data. The particular set of tools included Announcements, Schedule, Resources, Gradebook, Forums, and Site Info. Our average and worst response-time goals were 2.5 seconds and 30 seconds, respectively, and we measured our response times until the last byte transferred back to the client. We also wanted both the web server and database server not to exceed 75% utilization under a 500 concurrent-user load and an error rate less than 2% (responses other than HTTP 200). Our environment consisted of an 8-CPU Solaris server for the database, a 4-CPU Linux server for the application, several desktop machines to run Silk Performer, and a dedicated LAN with one switch to keep the network as small a factor in the testing as possible.

The test plan document, details the click paths our test students and test instructors followed during the performance tests. Here is a quick overview of the click paths.

Student Click Path	Instructor Click Path
login read announcements view schedule add event to personal site do readings upload resource to personal site preview questions do assessments utilize feedback review results communicate in forums communicate in email join worksite populate student profile search for information change e-mail preference browse help logout	login post announcements upload roster upload resources create assignments preview questions review grades export grades import grades add schedule event change role add tools to worksite edit site information search for information logout

Sakai Distro

The distro we used is based on the 2-5-x "Cafe" distro which is theoretically the minimum source code required to successfully build and start up Sakai. (https://source.sakaiproject.org/svn/cafe/branches/2-5-x/).

The actual externals defining the source code distribution under test are attached to this page. Implementation notes:

- We added the course management default implementation in order to access database-persisted AcademicSessions.
- We added the calendar module to meet Wiley's usage pattern requirements. This introduced a dependency on the assignments module.
- We added the gradebook module to meet Wiley's usage pattern requirements. This introduced a dependency on the sections module.
- We added the search module to meet Wiley's usage pattern requirements. This introduced a dependency on the mailarchive module.
- We added the msgcntr module to meet Wiley's usage pattern requirements. This introduced a dependency on the profile module which introduced a dependency on the privacy module.
- We switched to the "vanilla" (rather than Cafe) version of the user module to meet Wiley's usage pattern requirements for the preferences tool. This introduced a dependency on the syllabus module.
- We added a top-level "shared-deployer" module to deploy JSTL for a number of tools. (See SAK-12798\).
- We added the web module to fix the default Home page layout.

Data Management

We modified Alan Berg's provisioning scripts (see SAK-13352) to populate a large amount of data into the database (minor modifications compared to the work he's already put into the scripts). Specifically, we added functions to populate announcements, gradebook assignments, forum topics, and forum posts. We also added the ability to choose a per-site or per-instance distribution of created users, and we added functionality to create/specify a home page for worksites. Finally, we modified both the user/worksite creation and resource creation to produce very predictable names like worksite1, worksite2, worksite1-student1, worksite1-student2, resource1.pdf, and resource2.zip. This made authoring the Silk-Performer scripts much easier. Also, the default SOAP timeouts in provisioning.pl were too short. We had to increase them to 10 hours because of the large quantity of data we were loading. Here are the config files we used for provisioning; vlw.txt, lw.txt, mw.txt, sw.txt. Once we had the data loaded, we used Oracle Data Pump to obtain and restore backups of the data once we had it loaded, and we would periodically restore the database from backup once our testing had sufficiently polluted the database.

Bugs Identified

The bugs that were identified are detailed in the Performance Testing Results but we would like to gather feedback to determine if anyone else in the community has encountered these issues and is working on a solution. These issues included:

1. Forums
    - SAK-13188 Messages & Forums Synoptic Tool / More performance improvements
    - SAK-11463 Slow queries from Forums
2. Memory Leak upon User Login
3. CPU Consuming JVM
    - SAK-8932 Runaway CPU use on app server
    - SAK-13386 High CPU from stuck threads
    - SAK-11098 Stuck threads use excessive CPU

User Scripts

We used Silk Performer for our testing, and we can make the scripts available upon request.

Test Results

With the exception of one needed index (on the SURROGATEKEY column of the MFR_MESSAGE_T), we found the database performed remarkably well, and we had virtually nothing to tune on that end. However, we did find that the connection pool was recycling connections very rapidly, and that was causing us some real headaches. The problem proved to be that while we had the *minIdle* parameter set, we did not have *maxActive* specified. As soon as Sakai opened a connection, it would close it. We had a fall-off-the-cliff effect on performance as soon as active connections exceeded the *minIdle* setting.

We encountered performance issues with the forums tool, and we were unable to meet our stated performance goals there (average page-load time of 2.5 seconds). Rather than investigate and attempt to improve forum performance, we opted to delete the forum data and address the issues at a later date.

We discovered a memory-usage pattern associated with login that limits scalability and steady-state operation. Essentially, when a user logs into a particular JVM running Sakai for the first time, there are many objects created and, we assume, cached. We haven't identified the exact location of the problem, but we think those cached objects never age out of the cache. As new users login, the cache grows, and eventually the JVM runs out of memory. We had our JVM set for 1 gig of RAM, and we could achieve approximately 30,000 new logins before the server spent more time garbage collecting than it did processing. Adding memory to the JVM mitigates the issue, so we will probably choose to run larger JVMs and implement operational procedures (periodic server restarts) rather than attempt to modify the caching behavior,

On several occasions, we experienced stuck threads or threads that would run continuously and consume 100% of a CPU. We found a number of Jira tickets describing similar behavior (SAK-8932, SAK-13386, SAK-11098). These stuck threads skewed the metrics we gathered using sar, but we didn't really notice a significant effect on response times.

If you look at the response times graph in our performance test results, you'll notice that response times remained pretty even as we increased the number of users. In fact, the response times are almost the same with 500 users as with 1 user.