WileyPLUS E5

Load/Stress Test Results

Final Report

Version 1.0

Author: Cris J. Holdorph

Unicon, Inc.

Audit Trail:

Date	Version	Name	Comment
June 12, 2008	1.0	Cris J. Holdorph Alex Bragg	Initial Revision

Table of Contents

1. Reference Documents

- E5 Performance Scalability Goals.xls
- E5_Vanilla_Sakai_Test_Plan_v1_2.doc
- E5_Vanilla_Sakai_Test_Results_Data.xls

2. Summary

Performance testing was conducted on Sakai 2.5.x (revision was from the branch between 2.5.0 and 2.5.1) for the purposes of determining the baseline performance of Sakai. Testing was done with a basic set of tools configured in each worksite. The system was loaded with over 92,000 users and a series of announcements, resources and gradebook entries. Concurrent user testing began with a small number of users and gradually increased to support more and more users. This process also helped to debug the test environment itself, fixing errors in configuration and fine tuning the number of database connections, Apache HTTP workers and AJP processors.

The final results do not show any test meeting the goals established in the ***E5 Vanilla Sakai Test Plan*** document. However, the results also show no dramatic decrease in performance as more users are added to the test. As a result, 480 concurrent users perform almost as well as 15 concurrent users, given the final tests that were run.

In addition to debugging the test environment and gathering the initial test results, three bugs may have been uncovered. These bugs would most likely not have been visible in normal functional testing. Therefore performance testing can claim an additional measure of success in discovering bugs that might, otherwise, have gone unnoticed.

3. Environment Tuning and Debugging

The Vanilla Sakai environment was created following standard Sakai build procedures, modified only slightly to accommodate the Covalent distribution of Tomcat on the E5QA02 box.

During the course of baseline performance testing of the E5 release of WileyPLUS, we utilized two database servers, qa2.wiley.com and qa3.wiley.com. The first served us during initial performance testing, and the second served us for the remainder of our baseline testing. Both machines are 8-CPU, Sun-Solaris servers with 1800 MHz SPARC processors. The database's files were located in SAN-provisioned Veritas volumes. The network bandwidth between the application server, e5qa02.wiley.com, and the database was 1 gigabit maximum. We did not attempt to benchmark raw performance of the test machines nor the network because we perceived little or no value in doing so at the time.

On the database server, we utilized Oracle Statspack and the Solaris sar (/lib/sa/sadc) utility to gather statistics from the database and the server. The sar utility provided us with kernel-level statistics such as CPU idle times, CPU user times, CPU system times, network I/O metrics, and disk I/O metrics. The Oracle Statspack provided us with detailed information about query performance, wait events, and general instance performance.

Database Connections

Close monitoring of session activity showed that the database connection pool was recycling connections excessively. Ultimately, we found that we needed to specify both the *minIdle* and *maxActive* parameters in order to control the size of the connection pool. At first, we attempted to set the values to reasonable numbers. However, we still ran into problems. In order to make progress during the tests, we raised the number of connections to very large amounts, 250 *minIdle* (pre-allocated connections) with a *maxActive* number of 980.

Log Tuning

In order to debug the test environment it became necessary to change the logging levels of the Sakai system. Changing logging levels is done in the ${sakai.home}/sakai.properties file. Here is a sample of what we used for a portion of our testing.

```
log.config.count=4
log.config.1=ALL.org.sakaiproject.log.impl
log.config.2=OFF.org.sakaiproject
log.config.3=DEBUG.org.sakaiproject.db.impl
log.config.4=INFO.org.sakaiproject.component.app
```

Eventually we commented these lines out, because we no longer needed the extra logging information. While working on these logging levels, we also discovered 3 java .jar files were in an incorrect location. These files prevented logging from working normally. Once these files were removed, logging worked again as expected.

Search Index Repair

During the test script development, search was reported to not be functioning. Looking at the log files, there were errors being reported about unable to find or update the search index. We determined that the search index had probably been corrupted when by two things; 1) the disk space the search index used was filling up and 2) reloading the database. We decided to rebuild the search index. On the Sakai server, E5QA02, we moved the search index directories to a new partition with more space. We then removed the contents of these directories.

```
${sakai.home}/indexwork
${sakai.home}/searchjournal
```

We also needed to reset the database tables related to search to their initial state (to match the directory trees we removed). We deleted the following tables and allowed a Sakai restart to recreate these tables (this requires the property auto.dll=true in ${sakai.home}/sakai.properties):

```
SEARCHBUILDERITEM
SEARCHWRITERLOCK
SEARCH_JOURNAL
SEARCH_NODE_STATUS
SEARCH_TRANSACTION
SEARCH_SEGMENTS
```

After this, we logged on to the Sakai system administrator and used the Sakai User Interface to force search to rebuild the search index. After this process was complete, we took a new snapshot of the database and also took a corresponding snapshot of the directories mentioned above. Any time we rolled back the database, we also rolled back these search index directories to their previous state that corresponded to the database snapshot.

Removal of Forums

Early results in the performance test runs showed that the forums data was having a very negative impact on the test environment. Doing investigation in the Sakai community it was determined there could be bugs in the Sakai 2.5.x message center code that affects both the Forums and Messages tools. A decision was made to remove the data associated with these tools and remove the use of these tools from the test scripts.

To remove the forum data from the database we created backup tables of MFR_MEMBERSHIP_ITEM_T, MFR_MESSAGE_T, and MFR_TOPIC_T. We then deleted all records from MFR_MEMBERSHIP_ITEM_T that weren't associated with announcements. Next, we disabled foreign key constraints utilizing importing the primary keys for MFR_MESSAGE_T and MFR_TOPIC_T. Finally, we truncated those two tables, and re-enabled those constraints.

JVM Heap Size

Memory tuning is something every Sakai environment needs to do. Most tests were initially run with a JVM heap size of 1 gigabyte. This setting was increased to 2 gigabytes. The E5QA02 machine has 8 gigabytes of RAM and 4 processors. We felt increasing the default JVM heap size to 2 gigabytes was a realistic size given the RAM available in the box. These are the final JAVA_OPTS used during the last series of tests.

```
GC_OPTS="-XX:+UseParallelGC -
Xloggc:/opt/covalent/ers31/servers/sakai/logs/gc.log -verbose:gc -
XX:+PrintGCTimeStamps -XX:+PrintGCDetails -XX:+UseParallelOldGC"

JAVA_OPTS="-server -Xms2048m -Xmx2048m -XX:NewSize=1024m -
XX:MaxNewSize=1024m -server -XX:PermSize=256m -XX:MaxPermSize=256m  -
XX:+JavaMonitorsInStackTrace -XX:ThreadStackSize=256 $GC_OPTS"
```

New Logins / Returning Logins

While debugging the performance test environment, one theory we proposed, was maybe there was a difference between new user login and returning user logins. We tested this theory by turning up the Sakai logging for database component and measuring the use of database connections for the two different scenarios. It was difficult to measure precise numbers, due to background Sakai processes that operate during the test. However, it was clear that new user logins were using a lot more database connections. The numbers were somewhere in the range of 50 for returning users compared to 200 for new logins. We decided to do a massive login of as many users as possible and use returning user logins for subsequent tests.

New Database Index

Initial testing revealed the need for an index on the **SURROGATEKEY** column of the **MFR_MESSAGE_T** table. We created a simple binary tree index on that table column, and we had no further problems with queries on that table. In fact, the Oracle Statspack reports did not indicate any other problem queries.

Changed Network Access to QA3

Finally, the Statspack reports indicated an abnormally high wait time sending data back to the application server. The network team provided some assistance, and, at their suggestion, we moved the database server onto the same network as the application server. This was done by changing the way E5QA02 accessed the QA3 database. The access was changed to a different IP address. By accessing QA3, at least one router was eliminated from the network path between E5QA02 and QA3. This change did not have a noticeable impact on the test results.

4. Bugs Discovered

A few bugs were discovered during the test process. These bugs will need to be addressed before E5 goes into production.

Forums

In the very first tests, forums-related pages were the worst performing page requests in the system. The test plan had called for setting up a large number of sites and each of those sites had a significant set of forum data. The more users present in a site, the more data was present in forums for that site. This large amount of forum data seemed to cause the performance tests to struggle.

Around this same time discussion was taking place in the Sakai community about performance issues with this tool in this release of Sakai (2.5.x). The decision was made to remove forums from the test sequence, because it was not going to be used for the initial release.

There are at least two issues mentioned in the Sakai community Jira system that relate to this bug. The Jira issue numbers and titles are:

SAK-13188	Messages & Forums Synoptic Tool / More performance improvements
SAK-11463	Slow queries from Forums

Memory Leak

While we were performing the massive login/logout test to support returning logins (see previous section "New Logins / Returning Logins") we discovered that Sakai was running out of memory. As the test progressed, full garbage collections were occurring more often and recovering less and less memory. The test eventually started thrashing with full garbage collections occurring every few seconds and taking several seconds to run. The test was effectively brought to a standstill after only a few hours.

This same kind of test was repeated in another environment with similar results. The test was altered to then run returning user logins only, again cycling through a large number of usernames (over 92,000). This test also ran out of memory, it simply took longer to get to the same point.

Finally, the set of usernames was kept to a small list of only 1200 names. This test did not appear to run out of memory and was able to run for as long as the test was kept active. This suggests there is a memory leak related to the user login process. No Sakai community Jira issue was found to appear related to this issue.

CPU Consuming JVM

Several test runs exhibited strange behavior with the CPU usage spiking on the application server. After one test was ended, we noticed that the Sakai JVM was still taking a large amount of CPU. We confirmed that nothing was hitting the server; however the system was still 'busy'. We then investigated the CPU numbers reported by the SAR utility. It appeared that at some point during the test the CPU spiked significantly and then never went back down to normal levels. When this problem was first noticed, we were able to repeat the test without changing any variables and the problem would sometimes go away. During the final series of tests, the problem manifested itself during 240 user test and all subsequent tests after that.

This appears to be a bug in Sakai. We have very little information at this time. One primary source of information would typically be a stack trace of all running threads in the JVM. This information can be gathered by sending a *kill -3* signal to the JVM process. When this was attempted on the E5QA02 machine, we did not receive the expected output in the tomcat log file. We are unsure why this standard technique does not work in this environment.

There are a few issues mentioned in the Sakai community Jira system that might be related to this bug. The Jira issue numbers and titles are:

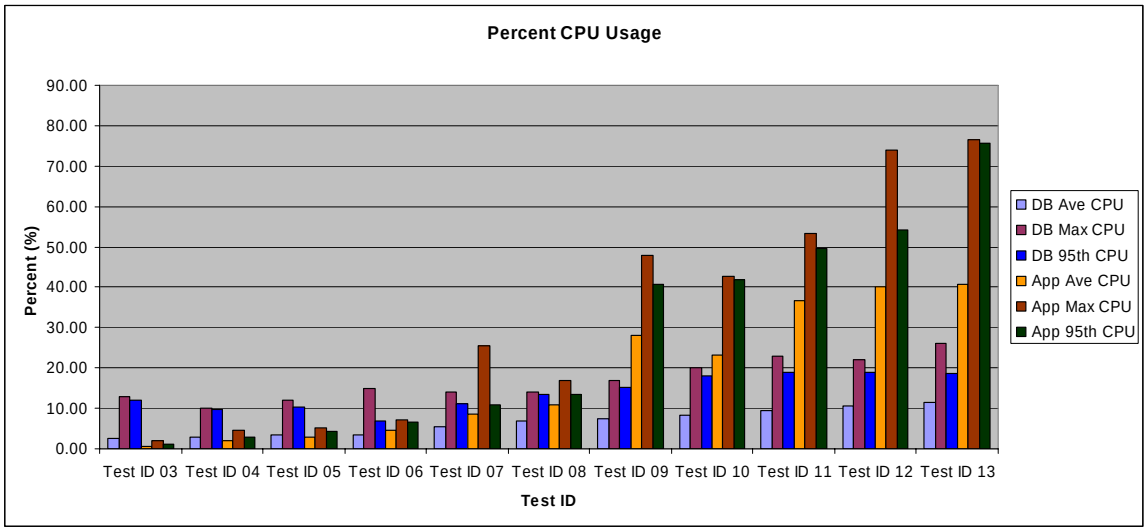
SAK-8932	Runaway CPU use on app server
SAK-13386	High CPU from stuck threads
SAK-11098	Stuck threads use excessive CPU

5. Test Results Analysis

When performance testing began, the results were horrible. The environment needed a lot of tuning. After we had finished tuning the environment some, we were able to start getting all the

tests to succeed (no errors reported by the testing tool). Then we started to get results. These results have been recorded in the referenced Excel spreadsheet.

During the final phase of testing, we completed a total of 10 tests. Each subsequent test increased both the number of concurrent users and the overall length of the test. The following graph shows the CPU usage on both the application server and the database server.



The graph shows us a couple of interesting things. First, the database does not seem to shoulder a particularly heavy load compared to the application server, which leaves plenty of room for scaling the application layer. Second, we can see from the smallest tests that there is some background load running on the database server that may contribute significantly to the CPU metrics we gather there. Finally, the average CPU load across all the tests is another indicator that we have plenty of room to scale load before we require more database hardware.

Overall, the application's usage of the database appears to be efficient and well tuned. The Oracle Statspack reports show no significant contention on cache buffer chains, which is one of the primary indicators of queries with poor execution plans. Also, the reports show little or no latch contention, no unnecessary sorting, and no alarming memory growth. In most cases the Statspack reports indicate no real room for improvement. However, the reports do show a few metrics that are probably contributing to a marginally longer average response time during performance testing.

Of the total execution time, CPU processing accounted for about 60-70% of the total time, time spent transmitting data to/from the application server accounted for 10-20%, time spent writing data to disk accounted for 10-15%, and time spent parsing queries accounted for about 5%. The CPU processing time and the time spent writing data to disk are both fairly healthy figures. However, network time and parsing time might both need some attention.

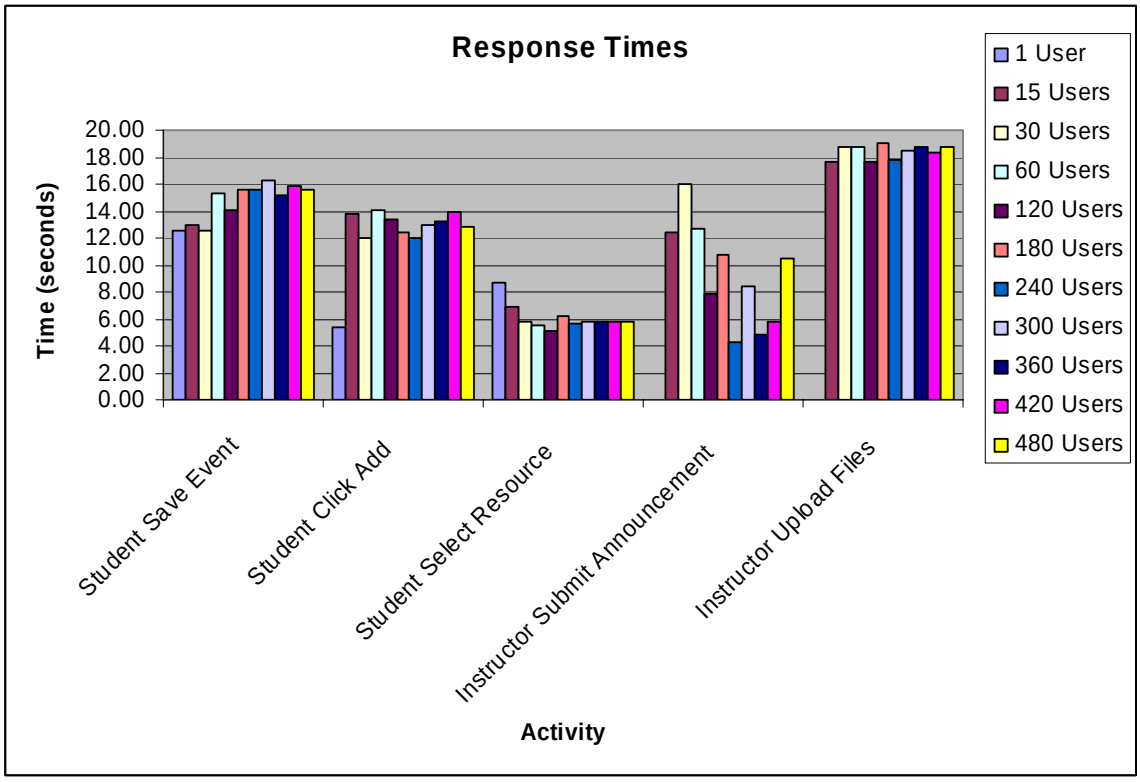
The Statspack reports show an extremely low execute-to-parse ratio. The application is preparing queries for execution (by having Oracle parse them and select an execution plan) but then executes no more than 5% of those prepared queries. In this testing soft parses accounted for more than 99.5% of those query parses, which is the least expensive parse possible. In a soft parse, Oracle executes a checksum on the submitted query string and generates a hash. Oracle then uses the hash to scan the library cache for the execution plan it might have previously generated for that query. If Oracle gets a hit from the library cache, that is a soft parse, and Oracle will re-use the previously generated execution plan. Otherwise, Oracle must generate a

new execution plan (a hard parse). Soft parses accounted for up to 5% of total execution time, and 95% of that time was unnecessary because of the low execute-to-parse ratio.

During initial testing, we noticed an inordinate amount of time spent transmitting data back to the application server. We switched to an interface running on the same network as the application server, but that seems to have had no effect. In one test (a typical example), Oracle received about 800 MB of data from the application server and returned 1500 MB over a one-hour test, which is far below the maximum bandwidth of the network (less than 1%). Oracle spent a total of 162 seconds transmitting that 1500 MB back to the application server. At 1 GB network speeds, that total time should be less than 20 seconds theoretically.

Finally, of the total CPU processing time, Oracle spent approximately 25% of that time executing queries that were not using bind variables (unparameterized queries). Analyzing the performance of these queries is difficult because typically the Statspack report does not show them directly. The report displays the queries that contributed most to the load during the period the report covers, and parameterized queries almost always fill the top spots. Thus, we must either infer the performance of unparameterized queries from the Statspack reports or expend a significant effort finding them through other analysis. The reports infer that we have no unparameterized queries performing poorly, so expending such effort does not seem warranted at this time.

The following table shows the response times gathered during the 10 consecutive tests for the each page request that took longer then 2.5 seconds.



The graph shows us the response times do not appear to dramatically increase as the number of concurrent users is increased. These responses are extremely slow and need to be worked on, but Sakai is not getting any worse in performance as the number of concurrent users increase.

6. Suggested Next Steps

Several tasks have been determined as next steps following this performance testing. These tasks are:

Track Down Known Bugs

It is recommended that each of the bugs discussed in Section 4 (Bugs Discovered) be investigated and fixed. Where Sakai Jira issues exist, the Jira Issues should be analyzed. Any patches associated with the Jira Issues should be tested. Where no Jira issue exists, a test case that can be abstracted for the Sakai community should be created and a new Sakai community Jira Issue should be created.

Investigate Slow Response Times

Every test run showed several pages with response times that were longer then the goal of *2.5 seconds or less*. During many of these tests, the CPU activity of both the database and application server were both very low. The long response times do not appear to be because of system load. Investigation should be taken to try to determine the reason for the long response times in the case of low system activity.

Conduct Similar Testing with More or Less Data

One plausible reason for the bugs and slow response times seen by this performance testing and not seen by others in the Sakai community could be related to the data that was preloaded before testing began. Additional testing could be conducted using the same environment and scripts, but changing the amount of data in the database. For example, if only 1200 users exist, instead of 92,000 users, what do the response times look like then? If there are 200,000 users instead of 92,000, what do the response times look like? In addition to the number of users, the other data that was loaded in the system, announcements, sites, gradebook and resources, should be reduced or increased in similar ways.

Complete Remaining Tests from the Original Performance Test Plan

The ***E5_Vanilla_Sakai_Test_Plan_v1_2.doc*** test plan defined several tests that were not completed during this cycle of performance testing. Section 5 of the document (Load/Stress Test Types and Schedules) defines a Capacity Test, Consistent Load Test, Single Function Stress Test, Baseline Test and Load Balancer Test. From this list, only the Capacity test was truly attempted. One 24 hour Consistent Load test was attempted at the end of this round of performance testing; however, the CPU bug identified above made it difficult to determine the success of this test. The remaining tests should be conducted.

Tune the Number of Database, Apache and AJP Connections

Many settings were increased beyond reasonable numbers during this performance test cycle. Each of these values should be tuned before any rollout to a production environment is made. The best way to tune these values is to first determine the baseline performance. Once the maximum number of concurrent users are known, several tests should be run to determine if the response times for this number and CPU activity for this number of concurrent users is consistent. If consistent results can be obtained, then each setting should be reduced to the smallest number possible, until the results of a performance test start to deviate from the expected values.

The number of database connections can be set with the following properties in the ${sakai.home}/sakai.properties file:

```
minIdle@javax.sql.BaseDataSource=300
initialSize@javax.sql.BaseDataSource=100
maxIdle@javax.sql.BaseDataSource=980
maxActive@javax.sql.BaseDataSource=980
```

The number of AJP processors can be set by changing the **maxProcessors** value in the ${tomcat.home}/conf/server.xml file:

```
<Connector port="8009"
  minProcessors="5" maxProcessors="500" acceptCount="100"
  enableLookups="false" redirectPort="8443" debug="0"
  connectionTimeout="800000"  protocol="AJP/1.3" URIEncoding="UTF-8" />
```

The number of Apache HTTPD workers can be set by changing the httpsd.conf file and modifiying the following property:

```
MaxClients          500
```

Database connections should be tuned first, followed by Tomcat AJP processors and finally Apache HTTPD workers. Each value should be set to allow for some future growth.

7. Conclusion

The performance testing of Vanilla Sakai for the E5 WileyPlus effort can certainly be thought of as a success. Familiarity with Sakai was gained by many people who had not previously worked with Sakai. The testing environment was debugged, and the skills gained during that process will help in future environments for both testing and production. The performance testing process also gained insight to the current performance profile of many areas of Sakai.

Overall, the application's use of the database is very efficient. With the exception of the issue of transmitting data back to the application server, the reports do not indicate any more low-hanging fruit, and additional tuning efforts would likely have a very low return-to-expense ratio. Even if we did solve the data-transfer issue, the effect on the overall performance averages would be small.

Finally, several bugs were uncovered that would not have been found by conventional functional testing. Future efforts should work to address the bugs discovered as well as further investigate the open issues identified by this performance test process.