



***The Workshop on Performance and Reliability 9:
Pushing the Boundaries of Performance Testing Tools***

Tool Boundaries Avoided: Context for Choosing Custom Tools

Document Version 0.8.6

Last Updated: September 5th, 2007

Eric Proegler

ericp@onbase.com

Office of the CTO

Hyland Software, Inc.

28500 Clemens Road

Westlake, Ohio 44145

Ph: (440) 788-5000

Fax: (440) 788-5100

www.onbase.com

COPYRIGHT AND DISCLAIMERS

© 2007 Eric Proegler and Hyland Software, Inc.

Permission for use and publishing of this document by the Workshop on Performance and Reliability (WOPR) is granted. All other rights reserved.

Information in this document is subject to change without notice and does not represent a commitment on the part of Hyland Software, Inc. The software described in this document is furnished under a license agreement or nondisclosure agreement and may be used or copied only according to the terms of this agreement. Should you have any questions pertaining to discrepancies in this document, please contact Hyland Software, Inc.

Depending on the modules licensed, the OnBase[®] Information Management System software may contain portions of: Imaging technology, Copyright © Snowbound Software Corporation; CD-R technology, Copyright © Sonic Solutions; CD-R technology, Copyright © Rimage Corporation; OCR technology © ScanSoft, Inc; Mail interface technology © Intuitive Data Solutions; Electronic signature technology, Copyright © Silanis Technology, Inc.; Full text search technology, Copyright © Microsoft Corporation; Full Text Indexing technology: © Verity, Inc.; SYBASE Adaptive Server Anywhere Desktop Runtime, Copyright © SYBASE, Inc., Portions, Copyright © Rational Systems, Inc.; ISIS technology, Copyright © Pixel Translations, a division of ActionPoint, Inc. Portions contained within are licensed by U.S. Patent Nos. 6,094,505; 5,768,416; 5,369,508 and 5,258,855.

Hyland Software[®] and OnBase[®] are registered trademarks of Hyland Software, Inc. Application Enabler[™] is an unregistered trademark of Hyland Software, Inc. All other trademarks, service marks, trade names and products of other companies are the property of their respective owners.

Intended Audience

Attendees of the Workshop on Performance and Reliability, or others examining performance testing from a context-driven perspective. It is expected that the reader has experience and familiarity with using automated testing tools to performance test software systems.

Abstract

This paper describes an experience; a context and confluence of circumstances where a custom tool was the right choice for a performance testing project. The hope is to identify considerations in using homegrown tools while describing a situation where they were successfully used and deployed. In looking at why this worked, the circumstances considered necessary for success are identified, and what might be learned.

Table of Contents

INTENDED AUDIENCE	2
ABSTRACT	2
TABLE OF CONTENTS	3
NARRATIVE	5
CONTEXT	6
COMMERCIAL LOAD TOOL LIMITATIONS	7
THE CUSTOM TOOL	10
SUCCESS FACTORS	12
APPENDIX A: REPORT FROM TESTING ENGAGEMENT	13
TABLE OF FIGURES	14
REPORT SUMMARY	16
INTERPRETING TEST RESULTS	17
MODELING WORKLOAD	17
RESOURCE CONSUMPTION AND PERFORMANCE CURVES	18
CONCLUSIONS AND PREDICTIONS	20
METHODOLOGY	21
EQUATIONS	21
TUNING	23
WEB SERVER TUNING	23
DATABASE SERVER TUNING	24
ENVIRONMENT SUMMARY	25
WEB SERVER SUMMARY	25
DATABASE SERVER SUMMARY	26
CLIENT SUMMARY	27
LOGIN PERFORMANCE	28
ANALYSIS	34
COLD RENDERING	35
RAW DATA	35
ANALYSIS	41
IMAGE RENDERING	42
RAW DATA	42
ANALYSIS	48
AFP RENDERING	49
RAW DATA	49
ANALYSIS	55

BANDWIDTH EFFECTS.....	56
REPORT APPENDIX A: ADDITIONAL RESOURCES	59
REPORT APPENDIX B: REFERENCES.....	60

Narrative

After WOPR8, I started putting the word out internally that my group would performance test and consult for services revenue. Our first engagement was arranged shortly afterwards. A customer asked for us to measure the capacity of the hardware they already had in place, running our web server software. We quoted a small job, with the intention of turning this project around at the pace we are accustomed to.

We needed to choose a tool for our testing during the quoting process, so that we could pass through licensing costs in the quote. The important thing for us was to test effectively in the two days we had available onsite; any tool problems that would have the effect of reducing our testing scope were unacceptable.

Well in advance of this testing engagement, we had become disenchanted with our commercial load tool. There is a full section of this paper detailing the challenges of the tool, but in short, the load tool is difficult and time-consuming to code scripts with, locks us into canned reports and graphing, and is not particularly robust or reliable itself as software. I got a quote for a short engagement, and it was several thousand dollars for less than a month.

Based on this, we were ready to choose something else for our project. We considered other commercial tools, and OpenSTA. We also considered a custom tool we've been working on for a while. We started developing this custom tool to do tests that we couldn't perform with our commercial tool, because of licensing and reliability limitations. We have been working with it for several months, and though it was still evolving, it had been reliable in duration testing and high load situations.

In the end, we choose to use the custom tool for the project. The tool developer would be onsite with me, and I was confident or foolish enough to think that we could iron out any difficulties during the engagement.

The customer let the quote sit for several weeks, then suddenly wanted us onsite the following Monday. The engagement was scheduled for two days onsite, with two days of setup, analysis, and follow-up budgeted. A total of eight days of services was billed for both of us, and is pretty close to actual usage.

When we arrived onsite for the first day, the LDAP integration in our web server software required updating of the tool. The tool developer worked on the tool and scripts, and the tool and scripts were ready by 5 o'clock, allowing us to break for dinner.

I had made it a requirement of our engagement that we would be able to test after hours as we saw fit. We came back at 7pm, and started running tests and updating scripts. We ended up finishing by 1am with the small set of tests we had planned, verified our data capture, and packed up. The next day, we validated a load balancing switch, performed a hyperthreading test, helped with application-specific issues, delivered an oral summary of results, and then went home that night.

A week later, we delivered the report found later in this paper. We did some interesting resource cost-based analysis, and took an approach in providing capacity calculations that may be of interest, but this is beyond the scope of WOPR 9's theme. The customer was quite happy, so we can claim custom tool use on this project to be a success.

Having done this successfully, it is very apparent to me that this is not something to be undertaken lightly or casually. Without the work put in prior to the engagement on the tool, the availability of the programmer onsite, and some luck, this would not have been successful.

Context

I work for a software development firm that develops an ECM solution named OnBase, developed in C++/C#/ASP.Net, using Windows and IIS with a variety of database and storage platforms. Revenue nears nine figures, and development/quality assurance staff is around 150. Development is aggressive Agile, with an impressive number of features added between semi-annual major releases, with service packs being distributed year-round.

Approximately 6600 customers have been installed by the company's services group, channel partners, and OEM re-labelers. Imaging, parsing and import processing, workflow, a thick client, a web server and web services, and exposed APIs are all components of the software of concern to us.

Database/web/application servers, storage hardware, networks, custom middleware, and widely varying line of business uses are the context in deployment.

I lead a team of three that serves a number of purposes. We report to a senior manager with 35 years of experience, an essential architect of the software that now serves as the CTO's lead technical resource, guiding us and others. For now, I call us the Performance Team, though I'm taking impressive name suggestions.

We performance test software under development; we work with Development to isolate and fix bugs in areas such as thread safety, in addition to identifying unnecessary resource allocation and consumption in the software. We also provide troubleshooting and analysis in escalated support situations (frequently leading to the previous activity), evaluate technologies and their deployments with our software, present training to sales engineers, teach at our vendor conferences, and in the situation described here, earn services revenue through performance testing services.

This broad portfolio requires executing quickly and simply. We must choose a reasonable scope and stick to it. We generally first perform a small number of uncomplicated experiments with thorough instrumentation. We then apply rapid heuristics and group analysis, report, and then test further into areas of interest. We report written, orally, and/or through change requests. Methodology and conclusions may be challenged, so we need to show data, but documentation is generally quite light and often not more than email and a couple spreadsheets.

This pace is kept successfully because we have an accumulated foundation of knowledge concerning the software, supporting technologies and test lab hardware. Mentorship is readily available, intermediate levels of confidence in results can be expressed, mistakes are tolerated, and we are permitted to simply state conclusions and move quickly to the next task.

We definitely make trade-offs for this agility. We load test only some of the time, reducing our opportunity to become truly familiar with our tools. We do not have resources and time to invest in making our toolbox as robust and repeatable as it could be. The commercial load tool experience we have is significantly affected by this. We do not reach very high levels of justified confidence in our results, as we generally do not have the time to aggressively challenge conclusions from more than one or two directions. We live the 80/20 proportion, counting on getting enough right quickly enough to make up for the overall success percentage reflected in moving at this speed.

All in all, the fast pace and diverse focus makes for a high level of challenge and freshness, and fits our team very well. The frequent changes of priority, lack of closure, and uncertainty means that it is probably not a good fit for everyone.

The last point of context significant for this paper is a bias in the organization against open-source software and using it. I have not drilled to the bottom of it, but been snapped at loudly enough for asking questions that I tread lightly when talking about OpenSTA, for example.

Commercial Load Tool Limitations

Our commercial tool was chosen by Development and QA Management several years ago as a result of vendor familiarity. We trialed Compuware's QALoad at a time when our web browser software was .asp based, and used it very effectively at that time. We did not evaluate any other packages at that time, and management made the purchasing decision. We continued to use it effectively for several months. When our software moved to ASP.Net, the tool was not ready, and has not caught up yet.

Our context does mean that since we are not using the tool on a regular basis, we may be missing out on certain shortcuts or ways to save future work. On the other hand, the software we test changes often enough that we would be not getting a huge benefit from script reuse in any case, and we have certainly looked for and asked the vendor about the most troubling items.

Experiences with the tool described here refer to version 5.2 SP5 through 5.5 SP3. We have been in contact with our vendor, and after dropping WOPR names and so forth, we have an engaged vendor that seems to be trying to make things right. All the same, these criticisms are not hearsay, but hard experiences that the vendor has not challenged as we've reported and discussed them.

Our biggest challenge is the time it takes to execute even the simplest test. Coding load tests is time-consuming, and it is a good try when we can take a load script from recording to playback in less than half a day. This time is consumed primarily in coding scripts to accommodate capture/parameterization manually. This may or may not be reflective of what is expected by other load testers, but in our context, spending hours doing a repetitive task such as this is a speed-killer.

In our web server software, Global Unique identifiers (GUIDs) are used as result set handles, allowing them to be re-referenced. These handles are passed in an http response when a result set is generated. The commercial tool has no capability to catch this easily; we need to dig into raw responses and configure capture parameters from there. We then need to locate the appropriate places to push the GUID back into the URL and make those substitutions. In LoadRunner, this is a right-click operation to parameterize the data from the tree view, and then a prompt for global replacement.

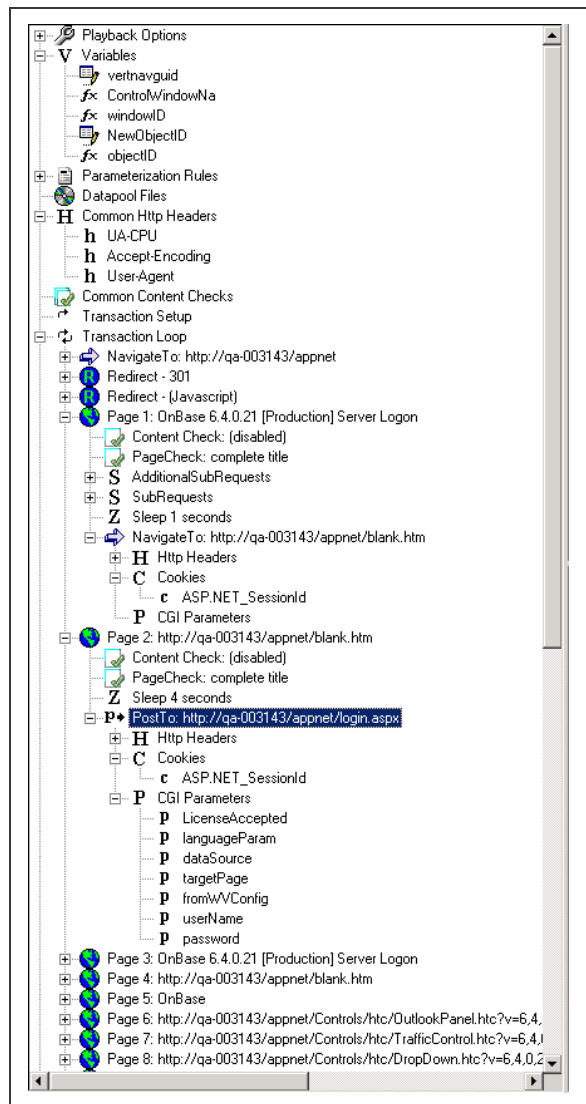
The other serious challenge in taking a load script from capture to replay is handling xml packages. Our load tool captures some xml responses, but not any inbound xml packages. Since query parameters (among other things) are often packaged this way, we need to configure these to simulate load against the web server properly. The load tool does not allow for capture-replay of these, so we must build xml payloads programmatically in string variables, and post them to the proper URL. In LoadRunner, again, this is all simple enough.

Handling these conditions requires manually editing the .c load script file generated by the load tool. Editing the load script file breaks the visual "tree" interface first presented by the tool. Most functions can be handled in the tree view, which is somewhat easier to navigate and work with than the script file.

Unfortunately, switching frames, constructing xml packages, and other activities cannot be done in the tree view, and any changes applied to the load script through the interface will trounce changes made directly to the script file underneath. This makes the interface dangerous at best, and of limited use, since we cannot put many activities (such as our manual captures) into context with the GUI.

Though we attempt to make the changes provided for by the interface before we go to the script file, if we miss or forget any, we will end up forking the test script to copy and paste back to the working copy we end up load testing with. That's all if we keep it straight and don't trounce any work we've already done with the load script.

Our challenges with our load tool don't stop once the load script is prepared. Conducting tests has its own set of problems. For example, attempting to capture Windows performance counters is very hit-or-



```

//----- REQUEST # 4 (see action item on Page 3) -----
// current page url is http://qa-003143/appnet/login.aspx
//
Set(NEXT_REQUEST_ONLY, HEADER, "Accept", "/*/*");
Set(NEXT_REQUEST_ONLY, HEADER, "Accept-Language", "en-us");

Set(NEXT_REQUEST_ONLY, CHECKPOINT_NAME, "Page 4 - blank.htm");

Navigate_To("http://qa-003143/appnet/blank.htm");
DO_SLEEP(5);

//----- REQUEST # 5 (see action item on Page 4) -----
// current page url is http://qa-003143/appnet/blank.htm
//
Set(NEXT_REQUEST_ONLY, HEADER, "Accept", "image/gif, image/x-xbitap, image/jpeg",
    "image/png, application/vnd.ms-excel, application/vnd.ms-powerpoint,
    application/msword, application/xaml+xml, application/vnd.ms-xpsdocument",
    "application/x-ms-xbap, application/x-ms-application, /*/*");

Set(NEXT_REQUEST_ONLY, HEADER, "Accept-Language", "en-us");

Set(NEXT_REQUEST_ONLY, POST_DATA, "LicenseAccepted", "null");
Set(NEXT_REQUEST_ONLY, POST_DATA, "languageParam", "en-us");
Set(NEXT_REQUEST_ONLY, POST_DATA, "dataSource", "NEWSW2");
Set(NEXT_REQUEST_ONLY, POST_DATA, "targetPage", "NavPanel.aspx");
Set(NEXT_REQUEST_ONLY, POST_DATA, "fromWVConfig", "False");
Set(NEXT_REQUEST_ONLY, POST_DATA, "userName", "abr46c");
Set(NEXT_REQUEST_ONLY, POST_DATA, "password", "abr46c");
Set(NEXT_REQUEST_ONLY, CHECKPOINT_NAME, "Page 5 - Logging In");

Post_To("http://qa-003143/appnet/login.aspx");
Verify(PAGE_TITLE, "OnBase");

// Extract the substring found between the two strings specified below.
//
vertnaviguid = Get(REPLY, STRING, ModifyEncoding(UTF8, "
CreateSessionManager().\r\n"
"\r\n"
"window.name = \""), ModifyEncoding(UTF8, "\r\n"));

DO_SLEEP(1);

//----- REQUEST # 6 (see action item on Page 5) -----
// current page url is http://qa-003143/appnet/login.aspx
//
Set(NEXT_REQUEST_ONLY, HEADER, "Accept", "/*/*");

Set(NEXT_REQUEST_ONLY, CGI_PARAMETER, "v", "6.4.0.21");
Set(NEXT_REQUEST_ONLY, CHECKPOINT_NAME, "Page 6 - OutlookPanel HTC");

Navigate_To("http://qa-003143/appnet/Controls/htc/OutlookPanel.htc");

//----- REQUEST # 7 (see action item on Page 6) -----
// current page url is http://qa-003143/appnet/Controls/htc/OutlookPanel.htc?v=
Set(NEXT_REQUEST_ONLY, HEADER, "Accept", "/*/*");

```

Load Tool Tree-view vs. Script

miss, and it is not rare that we simply stop capturing counters during a test, or can't configure them for capture in the first place.

The interface for choosing counters for capture is not easy to handle either, as a tree view is presented with each instance grouped. So, if you wanted to choose six process-level counters, you would have to expand a tree under each counter to find and select the instance of the process for each counter.

Because of the load tool's closed design, we have to successfully capture performance counters inside the tool, or they are not available for correlation in the same reports/graphs. We typically capture performance data and do our own eyeball comparisons to response times/user loads later when this is important.

Another problem: when conducting a load test, the load players leak memory. This ensures that we cannot run any kind of duration test, because the load player won't stay online long enough. We've shown this issue within a few hours of testing a single load script against a single web server.

This is still preferable to load test threads, and even the load controller crashing during a test, invalidating large sections of time, requiring state re-initialization, test restarts, and so forth. Threads crashing will report a message such as "Middleware Exception Error" to the controller interface, and then that vuser simply stops working. We can't figure out what this is beyond some failure inside the load software, as no meaningful debugging information is available.

We have found load clients to be more reliable when vusers are configured process-based instead of thread-based. This means that a new player process is started on the load client for each virtual user, preventing a single thread from blocking all the rest if it crashes at the wrong point. Unfortunately, this limits us to 10 vusers at most on P4 desktops with XP and 512MB of RAM. This means that our experience with the number of simulated users supportable versus the vendor's claims involves differences in multiple orders of magnitude. Even with thread-based vusers, we have never been able to run 100 vusers on a single workstation, let alone the 875 the tool estimates we can, or the large numbers quoted elsewhere in documentation and by vendor personnel.

This would be more of a concern if we had more than our current 100 vuser licenses. We also have a single license for conducting load tests, and can't run more than one test at a time. We could increase this with an additional investment, but we have reservations. We pay a five figure annual maintenance bill, and adding more virtual users and conductor licenses would be a significant cost, particularly for a tool we use only occasionally.

Not only do we have questions about reliability and usefulness, the most damning flaw is that the data collected by the load tool and the tool itself is not extensible. There is no API to allow us to build or extend missing pieces (extraction, for example) into the tool. We cannot fix what we don't like about the tool, address our largest time sinks, or make our own correlations and graphs.

There is no way to use the data collected by the load tool in any other program without time-consuming manual and unsupported effort. We did figure out how to extract from the results recording format, but there is significant mapping to be undertaken, and multi-step massaging to get the data into a format we can work with.

All in all, we haven't thrown the tool away yet, but that has a lot to do with available options, too. I did get maintenance paid this year, and anticipate doing it again in 2008, but I would like to have some better choices. With the progress of our homegrown tool, this may be possible at some future date. Or perhaps the tool will improve, and we will like it better. And of course, if another vendor were to offer a reasonably priced tool with the features we need, we'd have to consider that.

The Custom Tool

After experiencing a number of the limits imposed by our commercial tool, we started thinking about alternatives. In a software shop like ours, statements like “I could write that in a week” are often heard. A gifted young member of my team started working on the tool some time ago, and after many months, we have a functioning load driver.

Based on the experience with the commercial tool, we had particular requirements. We did not look for any sort of graphing or reporting features, as we found what was available in the commercial tools limiting, and wanted to do analysis and graphing in Excel anyways. Our requirements, pretty much in order:

- Make a load generator and response time capture program for our web client. Not a graphing package, not a performance counter collector. Provide meaningful operator feedback in terms of user and response time metrics.
- Make it reliable. It must be leak-free, and capable of operating for months unattended. Capture meaningful information about failures when they do happen.
- Be multithreaded, allowing a high number of virtual users, and scaling as additional processor resources become available. Be capable of simulating and maintaining thousands of user sessions.
- Make scripts modular, so that components can be assembled quickly into load scripts. The test for success will be whether we can prepare load tests significantly faster than with our current load tool.
- Accept input files for variables.
- Extensible results data that can be stored long term, backed up, and referenced easily. We knew right away that a database was the results medium.

Our toolsmith delivered (mostly, so far) on these requirements. We’ve run duration tests beyond 30 days, simulated 3000 users on our web server, and now have successfully completed a custom engagement with it. We are pretty happy with what we can do with the load tool, but more development and documentation is needed to make it faster for all of the team to operate.

The toolsmith describes his efforts this way:

Over the past few months, I have created a tool that functions with reusable blocks of scripting to rapidly develop tests against an OnBase Web Server. Although it is still under development towards its final incarnation, the current tool can easily create loads both long and short term.

Built on the .NET 2.0 platform, this tool takes advantage of innate memory allocation and management that makes it very stable memory wise. In addition, each virtual user is self contained within its own thread

with exception handling and restart ability, meaning that a VU will always be able to function, no matter what the error.

The scripting language is C# with a built-in structure and Web Session library that has been built and tested over time. This library automatically handles cookies, posts, gets, timeouts, etc, that occur with HTTP communications. The scripts are dynamically compiled and integrated into the testing tool, allowing for quick and easy changes. Future improvements will include easier editing and possibly dynamic debugging.

After compiling a user script, which contains a unique GUID for identification, the user may create user profiles using the script blocks they created. This process uses an easy Tree view with dynamic variables for parameterization. These variables can also be made into lists of values. Check marks are used in the list in order to indicate the loop(s) and action(s) of interest for display during the run.

Logging is accomplished via a table in a SQL2k5 database. The logging is detailed, and includes sleep times, action updates, sub-action updates (individual pages), errors, and more. This data can be easily queried and aggregated using native SQL queries, or exported for analysis through Excel or any other analysis package. The table contains a fixed postfix that shows the date and time the study was started. The table prefix is determined by the user during configuration.

The main screen of the tool allows individual adjustment of the user profiles along with metadata of the test that can be saved. The users and entire test can be saved as an XML file that is easily human readable and editable.

The next large extension to this platform is a distributed client piece that will allow load clients to be placed on multiple machines in order to create large loads that one client would not be able handle.

Success Factors

The first and most crucial success factor was having a skilled toolsmith that could create and adapt the tool to our circumstances and make it work. The value proposition of a custom tool is that it can coded to work a particular way; it follows that sufficient skill and ingenuity is needed to build and adapt the tool to the circumstances rapidly enough to be useful. It was critical that he was onsite to make changes and fixes, and that I was confident he would be able to do so.

Nearly as important was the time invested in developing and refining the custom tool prior to the engagement. It existed for months before the engagement came up. There was a certain amount of confidence in and experience with the custom tool needed to justify the risk in going on-site with it. It may not have been finished, but it had been used enough that it was possible to believe it could and would work for our project.

The level of frustration with our commercial tool is documented well here, but without a certain level of dissatisfaction in the first place, we would not have invested the time in developing the custom tool, or tried to use it on an engagement. The combination of this frustration and appropriate moxie/foolhardiness to believe we could iron out issues on site was needed to even try this.

Expectations with the customer were managed well enough that even when the tool was still being modified towards the end of the first day of the engagement, there was no panic or questioning of what we were up to. We kept the confidence of the customer high even when ours might have faltered.

Appendix A: Report from Testing Engagement

The remainder of the contents from here is the report as delivered to the customer that paid for it. Others collaborated on this report.

The customer name is changed for anonymity. Everything else is open season for discussion time, or follow-up questions before, during or after WOPR. Please contact Eric Proegler with any questions or comments.

Table of Figures

Figure 1 - Resource Consumption Curve Example	18
Figure 2 - Login Web Server Session.....	28
Figure 3 - Login Load Tool Reported Sessions.....	28
Figure 4 - Login Actions Attempted per Second.....	29
Figure 5 - Login Web Server Page Execution Time	30
Figure 6 - Login Load Tool Reported Response Time.....	30
Figure 7 - Login Web Server Available Memory	31
Figure 8 - Login Worker Process Private Memory	31
Figure 9 - Login Worker Process Virtual Memory	32
Figure 10 - Login Processor Utilization	32
Figure 11 - Login Context Switching.....	33
Figure 12 - Login Processor Queueing.....	33
Figure 13 - COLD Web Server Sessions	35
Figure 14 - COLD Reported Load Tool Sessions.....	36
Figure 15 - COLD Actions per Second	36
Figure 16 - COLD Page Execution Time	37
Figure 17 - COLD Action Response Times	37
Figure 18 - COLD Available Memory	38
Figure 19 - COLD Worker Process Private Bytes	38
Figure 20 - COLD Worker Process Virtual Bytes	39
Figure 21 - COLD CPU Utilization	40
Figure 22 - COLD CPU Context Switching.....	40
Figure 23 - COLD Processor Queueing.....	41
Figure 24 - Image Test Web Server Sessions	42
Figure 25 - Image Test Load Tool Sessions.....	43
Figure 26 - Image Test Load Tool Reported Actions per Second	43
Figure 27 - Image Test Web Page Execution Time.....	44
Figure 28 - Image Test Load Tool Action Response Time	44
Figure 29 - Image Test Web Server Available Memory	45
Figure 30 - Image Test Worker Process Private Bytes	46
Figure 31 - Image Testing Worker Process Virtual Bytes	46
Figure 32 - Image Test CPU Utilization	46
Figure 33 - Image Test Context Switching	47

Figure 34 – Image Test Processor Queueing.....	48
Figure 35 – AFP Web Server Sessions	49
Figure 36 – AFP Load Tool Recorded Session Count	50
Figure 37 – AFP Requested Actions per Second	50
Figure 38 – AFP Page Execution Time	51
Figure 39 – AFP Load Tool Recorded Response Time	51
Figure 40 – AFP Available Memory	52
Figure 41 – AFP Worker Process Private Memory	52
Figure 42 – AFP Worker Process Virtual Memory	53
Figure 43 – AFP Processor Utilization	54
Figure 44 – AFP Processor Context Switching	54
Figure 45 – AFP Processor Queueing	55
Figure 46 - Response Time Sweep of COLD Documents.....	56
Figure 47 - Response Time Sweep of Image Documents.....	57
Figure 48 - Response Time Sweep of AFP Documents	58

Report Summary

The OnBase Web Server is being deployed as part of the OnDemand/Content Manager replacement project at Central Insurance Mutual Insurance Company (CIC). Hyland Software's Performance Team was contracted to measure the capacity of the current web server installation. Testing was performed on-site July 30th and 31st.

This testing was designed to measure the capacity of a web server in terms of logging users on, and viewing COLD, AFP, and Image documents. Before the testing was conducted, the environment was tuned. This is described in more detail in the [Environment and Tuning](#) section of this report. Testing is described in the [Methodology](#) section.

Testing was performed against a single web server, and two load balanced web servers. In both cases, requests were directed to a URL serviced by an F5 content switch, <http://onbaseweb>. The test cases used were defined by CIC for acceptance testing. Other work has been done by CIC with these test cases to measure typical response times.

Response times and resource consumption were monitored and analyzed. A full discussion of results and analysis can be found in the [Conclusions and Predictions](#) section of this report. It is important to note that these results are an approximation depending on an extremely large number of factors. A change in hardware or software, however slight, might change the capacity of the system significantly.

The most significant conclusions are as follows:

Each CIC Web Server can support slightly more than eight units of work per second before increased response times due to resource consumption become unacceptable. At over ten units of work per second, the web server begins to issue errors due to resource exhaustion. These units are defined as 1 for retrieving and rendering an image, 1.14 for the activities for a COLD(text) document, 1.01 units for logging on to and logging off of the system, and 2.96 units for retrieving and rendering AFP documents.

Capacity for a web server is approximately 493 units per minute, or 29,752 per hour. A session that contains two views of AFP documents would have a cost of approximately seven units, one for the login and three per AFP view. This means that a single web server can accommodate nearly 4,400 of these sessions per hour. If the model session being considered viewed two image documents, the cost per session would be about three units, and the capacity would be approximately 9,900 sessions per hour.

The limiting factor in performing more activities on this web server is CPU availability. Higher performance would best be achieved with additional processor cores, allowing the web server to better manage CPU context switching under a load of many users.

Testing with load balancing in place showed with two web servers, twice the work could be performed with similar response times. Examining performance of the database server suggests that a third web server could be added to an otherwise unloaded database server with the same scalability characteristics. Adding a fourth web server would likely approach a threshold of requiring the database server to be scaled vertically. The consideration of additional work against the database server from the ARMS application, processing, and other work suggests that with a workload that would reach capacity requiring three web servers, this threshold would be reached earlier.

Further information on placing the results of this testing in context can be found in the [Context: Interpreting Test Results](#) section of this report.

Also found in this report is an analysis of the effects of WAN link speed on response time for these test cases. This can be found under [Bandwidth Effects](#).

Interpreting Test Results

Due to the combination of scheduling challenges, limited reporting from the current solution, and customer preference, this engagement was unusual in that no input for predicted workload was considered. At the customer's request, measurement of capacity in a generic sense was taken from exercising test cases chosen by CIC.

The results obtained are approximations based on testing specific hardware and software. As technologies and software change, these results could change considerably. These results reflect a particular experiment, designed to simulate certain activities. They do not account for usage of the entire OnBase solution, workload spikes, or other variables. These measurements are specifically of how many of a given activity can be performed on the current physical hardware with the current software.

These measurements focused on CPU availability at the web server, as this is typically the limiting resource for web servers generally and the OnBase Web Server specifically. This also was the case at CIC.

The included spreadsheet contains equations for examining server capacity based on activity thresholds. By plugging variables in, a predicted web server capacity in terms of CPU can be derived. Depending on the nature of activities against the web server and system as a whole, other bottlenecks may arise.

For example, the retrievals used in this example are selective. If wild cards were used for keywords, the size of the result set might cause larger response times, in addition to creating work that would not scale on the same basis.

Modeling Workload

Load Testing is equivalent to a DDOS (Distributed Denial of Service) attack. It is not difficult to break a web server by flooding it with more requests than it can handle. Applying the correct load is the biggest variable in the experiment.

Any load testing needs usage data to put it in context, conduct meaningful tests, and derive how many servers are needed, even if it is known how much usage a single server can support. If replacing an existing application, there should be logs from which to get usage data from, and project future growth in usage from. Business stakeholders may also have data about the level of activity they plan on supporting, and it may be more appropriate (and easier) to size to that.

Some basic questions to get started:

- How many sessions per day? Average? Peak?
- How many sessions per hour? Average? Peak?
- What activities will users of the system perform?
- How many instances of each activity are performed per session?
- How frequently are these activities performed?
- How long does a typical user spend executing activities?
- What is the business context of these activities?

- Are there any rushes to account for? Monday morning, Friday afternoon, end of month/quarter/year?

For example, consider a requirement of supporting 250 users. Many load testers would start here, record themselves quickly clicking through a well-practiced transaction to login, retrieve a document, view it, and logout. They might then simulate 250 virtual users executing one session per minute.

Looking more at the business process or production usage, it might be learned that less than 100 users are logged on at any one time, but are logged on for extended periods of time. Watching a user bring up a document and work with it in a separate line of business application, may reveal that there is a document retrieval every 15 minutes, with two documents viewed in that time.

The first scenario explores resource limitations to help with capacity planning, generating 15,000 sessions, retrievals, document views, and logouts an hour, or more than 4 per second of each. By applying this type of load, the system could be examined to see how it degrades under abuse. Anything this test reports about response times or reliability is dangerously misleading.

A scenario modeled on the production usage is a much better choice to verify an application's response times. To simulate production usage, transaction rates should be realistic. A scenario based on the data above might generate 100 sessions, 400 retrievals, 800 document views, and 100 logouts an hour. This test is a much better predictor of user experience and application reliability.

There are reasons to do each type of test, but it is crucial to know which type is being performed and why. The difference in load against the web server between these two tests is more than 40 times (4000%) the number of activities. Response times, resource availability, and reliability may suffer from the limitations imposed by a scenario that does an unrealistic amount of work, or one that isn't sized to the available hardware.

Resource Consumption and Performance Curves

When examining test results, the goal is to identify "the knee of the curve", or the point at which response times or resource consumption stops growing in a linear or logarithmic fashion, and begins to grow exponentially. Here is an example:

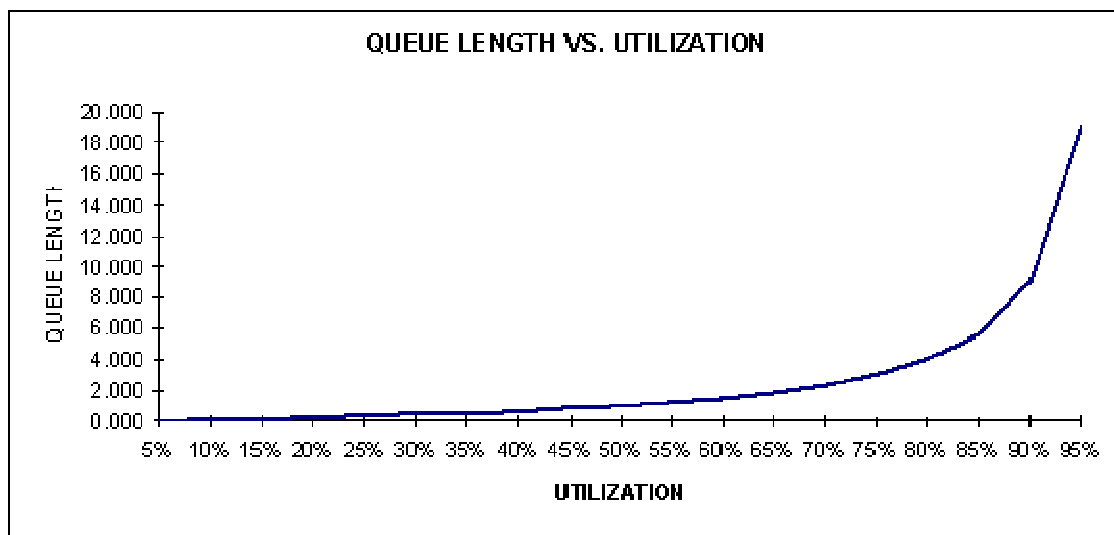


Figure 1 - Resource Consumption Curve Example

The increasing queue length will cause response times to be slower. In this case, queue length is a function of utilization. At approximately 70% utilization, the queue length begins to grow at an increased rate, and this rate of growth only increases as utilization climbs. At 80% and 85%, the curve can be seen

to increase even more dramatically. Perhaps at 75% utilization, the response times may have been acceptable, but they quickly become less so.

Every installed software system has a capacity limit, and a suitable operating range. Exceeding a system's capacity causes the software to stop functioning correctly. Exceeding the operating range is easiest to see by observing the increase in response times. Exceeding either capacity or operating range means just that; overloading resources will cause errors to occur, but it is important to assign proper causality to these.

It is a function of good software design to have capacity limited by available resources, rather than the software itself. Eventually, a piece of software will not run faster even with additional hardware resources. Hyland Software's Performance Team has simulated loads into the thousands of users, and found that as additional CPU capacity is provided, the web server continues to grow in capacity until other resource limits are encountered.

In the CIC testing, this was also the case. Here, additional CPU cores would increase capacity more than higher CPU speeds. Recommended practice is to scale web servers horizontally to deal with increased load, not to build more powerful web servers. There is also a redundancy/high availability value added from this type of deployment.

Conclusions and Predictions

These measurements describe a particular experiment, with a particular workload, in a particular environment. As components of the system change (software, hardware, network, etc.), or additional load is applied to the system and/or solution as a whole, results may change significantly. Though numbers of some precision are used for purposes of calculation, they are approximations based on specific test cases derived in a specific environment.

After examining the capabilities and capacity of the systems involved, it is believed that each of the in place web servers can handle a load of hundreds of active users or thousands of occasional users. After examining the costs of the common activities under test (logging on, retrieving and viewing images, retrieving and viewing COLD data, and retrieving and viewing AFP documents), an overall capacity was determined.

The following equation describes the capacity of each web server.

$$\frac{29,752}{1.03l + 1.14c + 1i + 2.96a} = \text{total users}$$

l = number of logins in one hour
c = number of cold documents retrieved in one hour
i = number of images retrieved in one hour
a = number of AFP document retrieved in one hour

For example, if it is expected that the average user will login twice, and retrieve and view 2 documents of each type per hour, you would plug in the numbers:

$$\frac{29,752}{1.03(2) + 1.14(2) + 1(2) + 2.96(2)} = \text{total users}$$

$$\frac{29,752}{2.06 + 2.28 + 2 + 5.92} = \text{total users}$$

$$\frac{29,752}{12.26} = 2427 \text{ sessions/hour}$$

An Excel 2007 file that will permit plugging in these variables to rapidly predict capacity is included with this report.

A round of testing with a second web server in place was performed. All activities showed a doubling of capacity without response time growth, indicating that the database and database server were scaling appropriately. Database server CPU did not exceed 20%, and other counters indicated acceptable performance with additional capacity available at this level.

Without the other OnBase activity in place for background, it is difficult to make a prediction regarding the need to scale the database server upward if additional web servers were deployed. The database server appears capable of comfortable serving the load currently in place.

Methodology

Load testing for the purpose of sizing a server or application works best in a controlled environment. This minimizes the number of variables and variation in the environment, making for an easier analysis of data. This level of isolation was provided during test runs by performing them off-hours, with other OnBase activities suspended during the test.

The most common way to establish a load against a server-based architecture is a multi-threaded and/or multi-computer simulation of user activity. This employs software that can send simultaneous requests to the server that mimic the communications seen from a live user.

There are many commercial and open source tools available to accomplish this task. Custom tools are also a choice, as they were used here. A partial list of commercial load tools:

- Borland® SilkPerformer® - <http://www.borland.com/us/products/silk/silkperformer/index.html>
- Compuware® QaLoad® - <http://www.compuware.com/products/qacenter/qaload.htm>
- Mercury® LoadRunner® - <http://www.mercury.com/us/products/performance-center/loadrunner/>

Open-source alternatives include

- OpenSTA - <http://www.opensta.org/>
- "The Grinder" - <http://grinder.sourceforge.net/>

During the course of performance testing at Hyland Software, a need was identified for a custom tool to exercise the OnBase Web Server. Though still under development, the tool successfully creates realistic load comparable to a live user and collects statistics related to its actions. It has successfully run long-term load tests with durations over one month, and simulated loads of thousands of users.

This tool was used on-site to provide load and collect data on server responses. Statistics were collected from other sources (database activity, performance monitoring) to confirm that proper load was being applied and to provide scientific validity.

A collection of Active Directory accounts was provided, and the load client tool used 250 separate accounts for diversified user context under load. Load was applied and increased gradually. By observing the resulting effects on resource utilization and remaining capacity, critical thresholds of activities were identified in terms of requests per second. In this context, requests is a user logic measurement, corresponding to a system activity, such as logging on, or viewing an image, COLD, or AFP document.

Equations

In testing particular functions, the following critical values of requests per second were found:

- Login – $8.06 = .124 \text{ sec / req}$
- COLD – $7.25 \text{ req/sec} = .138 \text{ sec / req}$
- Images – $8.25 = .121 \text{ sec / req}$
- AFP – $2.79 = .358 \text{ sec / req}$

The base assumption for this analysis is that a server is a finite resource that can be quantified. Using the above critical values, an equation for the calculation of maximum user load given a user profile can be derived.

First, the values must be normalized into units of work. Taking the smallest value of .121 sec / req, the rest of the values can be converted:

- Login - 1.03 units
- COLD – 1.14 units
- Images – 1 unit
- AFP – 2.96 units

To calculate the amount of units per hour a server can support, simple unit calculation is used

$$\left(\frac{.121 \text{ seconds}}{\text{unit}} \times \frac{1 \text{ hr}}{3600 \text{ sec}} \right)^{-1} = 29,752 \frac{\text{units}}{\text{hour}}$$

Using this result, an equation for the number of users supported by a web server can be derived. First, adding up the units used by one user per hour:

$$1.03l + 1.14c + 1i + 1.77a = u$$

Where

$$\begin{aligned} l &= \text{number of logins in one hour} \\ c &= \text{number of cold documents retrieved in one hour} \\ i &= \text{number of images retrieved in one hour} \\ a &= \text{number of AFP document retrieved in one hour} \\ u &= \text{total units used by user} \end{aligned}$$

This can now be plugged in and solved for an equation expressing the total number of users supported by a server, given the number of logins along with the number and type of documents retrieved in one hour by one user.

$$\frac{29,752}{1.03l + 1.14c + 1i + 2.96a} = \text{total users}$$

Tuning

Web Server Tuning

Microsoft .NET is installed, by default, with client connection settings. This means that a .NET application is limited in the amount of resources it is able to access. The OnBase Web Server relies on the .NET framework, and as such is limited by any settings applied to it.

While at CIC, the Performance Team tuned several settings on the Web Server in order to fully utilize all the resources available to it. Any other web servers added to the solution should also be tuned in a similar manner.

The following options are found in the *machine.config* file, located at `WINDOWS_ROOT\Microsoft.NET\Framework\v1.1.4322\Config` and are values that are recommended by Microsoft (Microsoft, 2004).

Max Connections – Maximum number of concurrent connections allowed to the server

Default: 2

Recommended: 12 * # of CPUs

Set To: 24

The `maxconnection` setting controls the maximum number of outgoing HTTP connections that a machine can accommodate. In the case of distributed web applications, this refers to the ASP.NET connection between the server and its clients. By increasing the number of connections, we increase the number of concurrent requests the server can handle.

MaxIOThreads – Maximum number of threads dedicated to IO operations

Default: 20

Recommended: 100

Set To: 100

The `maxIoThreads` setting controls the maximum number of I/O threads within the .NET thread pool, and are also considered to be "completion" threads. The number placed in its value is automatically multiplied by the number of processors within the machine.

MaxWorkerThreads – Maximum number of threads available for other tasks

Default: 20

Recommended: 100

Set To: 100

The `maxWorkerThreads` setting controls the maximum number of worker threads in the thread pool. Like the `maxIoThreads` setting, the number placed in this setting is automatically multiplied by the number of processors on the machine.

MinFreeThreads – Minimum number of threads available, even during idle time.

Default: 8

Recommended: 88 * # of CPUs

Set To: 176

The `minFreeThreads` setting is used as a low water mark indicating that the worker process is to queue all incoming requests when the number of available threads in the thread pool falls below `minFreeThreads`' value.

MinLocalRequestFreeThreads – Similar to above, but for local machine requests

Default: 4

Recommended: $76 * \# \text{ of CPUs}$

Set To: 152

The minLocalRequestFreeThreads setting is used as a low water mark indicating that the worker process is to queue requests from localhost when the number of available threads in the thread pool falls below minLocalRequestFreeThreads (when a web application makes a request to a local web service.)

Database Server Tuning

A number of steps were taken by Eric Proegler of Hyland Software to improve performance of the database server prior to the execution of tests. These are documented here, and were each shown to Chris Peer of CIC (SQL DBA).

Physical memory in the database server was increased to 8GB. SQL Server was configured to use 7.1GB of this RAM. This preserves physical memory for the operating system, and helps avoid excessive paging and overcommitted memory.

SQL Server's TempDB was previously configured with a single file sized to 1GB, with no autogrowth permitted. TempDB was changed to use four 4GB files on the TempDB volume, providing more space for sort operations and reducing logical contention. Autogrowth was not enabled.

Maintenance Plans that performed index statistics updates had been failing for several months. Though the job record showed that the maintenance plans completed successfully, this was erroneous. The maintenance plans failed when they ran out of space in the TempDB file. The sampling percentage was set to 10%, but Hyland recommends 100% sampling. After increasing the size and number of files in TempDB, and the sampling percentage, a full update of statistics was performed.

CPU affinity for SQL Server was set to use the first four virtual processors on the database server. Because the CPUs are hyperthreaded, the four CPUs present appear as eight to Windows. All eight CPUs were assigned to SQL Server.

Environment Summary

Data for the environment was collected with AIDA32 (Miklos, 2006) and summarized here.

Web Server Summary

Computer

Operating System	Microsoft Windows Server 2003, Standard Edition
OS Service Pack	Service Pack 2
Computer Name	SWEB200 (OnBase Web)
User Name	jmwtemp
Logon Domain	CICDOM1
Date / Time	2007-07-30 / 08:05

Motherboard

CPU Type	Dual Xeon, 3600 MHz (4.5 x 800) (HT)
Motherboard Name	Dell Computer Corporation PowerEdge 1850
System Memory	3072 MB
BIOS Type	Phoenix (10/03/06)

CPU Properties

CPU Type	Dual Intel® Xeon™ 3600 MHz (4.5 x 800) (HT)
Original Clock	3600 MHz
L1 Trace Cache	12K Instructions
L1 Data Cache	16 KB
L2 Cache	2 MB (On-Die, ATC, Full-Speed)

Storage

Disk Drive	PERC LD 0 PERCRAID SCSI Disk Device
Floppy Drive	Floppy disk drive
Optical Drive	TEAC CD-224E (24x CD-ROM)

[Drive #1 (33.9 GB)]

Partition	Partition Type	Drive	Start Offset	Partition Length
#1	Dell Utility		0 MB	31 MB
#2	(Active) NTFS	C:	31 MB	12291 MB
#3	NTFS	D: (Data)	12323 MB	12009 MB
#4	NTFS	E: (Logs)	24332 MB	10338 MB

Network

Network Adapter	TEAM : Team #0
Interface Type	Ethernet
Hardware Address	00-11-43-DD-29-4F
Connection Name	Local Area Connection 3
Connection Speed	1000 Mbps
MTU	1500 bytes
IP / Subnet Mask	10.210.2.57 / 255.255.0.0
Gateway	10.210.1.250
DNS	10.210.1.100

DNS	10.210.1.130
-----	--------------

Database Server Summary

Computer

Operating System	Microsoft Windows Server 2003, Enterprise Edition
OS Service Pack	Service Pack 2
Computer Name	SCIC068
User Name	jmwtemp
Logon Domain	CICDOM1
Date / Time	2007-07-30 / 09:22

Motherboard

CPU Type	Quad Xeon, 3333 MHz (5 x 667) (HT)
Motherboard Name	Dell Computer Corporation PowerEdge 6850
System Memory	8187 MB
BIOS Type	Phoenix (07/13/07)

CPU Properties

CPU Type	Quad Intel® Xeon™ MP, 3333 MHz (5 x 667) (HT)
Original Clock	3333 MHz
L1 Trace Cache	12K Instructions
L1 Data Cache	16 KB
L2 Cache	1 MB (On-Die, ATC, Full-Speed)

Storage

Drive	System	Serial	Total Size	Free(MB)	% Free
C: (SCIC068_C_Local_SystemFiles)	NTFS	F010-7CDF	34643 MB	19368	56 %
D: (SCIC068_D_Local_TRAN_LOGS)	NTFS	B6DF-68EB	34671 MB	34442	99 %
E: (S068_E_SAN_TEMPDB)	NTFS	B229-4C65	51222 MB	46380	91 %
F: (S068_F_SAN_DATABASES)	NTFS	E232-5543	153637 MB	92046	60 %
Z:					

Network

Network Adapter	BASP Virtual Adapter
Interface Type	Ethernet
Hardware Address	00-14-22-08-9D-91
Connection Name	Team 1
Connection Speed	1000 Mbps
MTU	1500 bytes
IP / Subnet Mask	10.210.1.68 / 255.255.0.0
Gateway	10.210.1.250
WINS	10.210.1.100
WINS	10.210.1.130
DNS	10.210.1.100
DNS	10.210.1.130

Client Summary

Computer

Operating System	Microsoft Windows XP Professional
OS Service Pack	Service Pack 2
Computer Name	DSTAVA7 (Dell Desktop Image 2)
User Name	jmwtemp
Logon Domain	CICDOM1
Date / Time	2007-07-30 / 09:46

Motherboard

CPU Type	Dual Unknown, 2800 MHz (3.5 x 800)
Motherboard Name	Dell Inc. OptiPlex GX620
System Memory	1014 MB
BIOS Type	Phoenix (03/31/06)

CPU Properties

CPU Type	Intel® Pentium® D CPU 2.80 GHz
Original Clock	2800 MHz
L1 Trace Cache	12K Instructions
L1 Data Cache	16 KB
L2 Cache	1 MB (On-Die, ATC, Full-Speed)

Storage

Disk Drive	HDS728080PLA380
Floppy Drive	Floppy disk drive
Optical Drive	HL-DT-ST CDRW/DVD GCC4482

[Drive #1 (74.5 GB)]

Partition	Partition Type	Drive	Start Offset	Partition Length
#1	(Active) NTFS	C: (DSTAVA7_C)	0 MB	76285 MB

Network

Network Adapter	Broadcom NetXtreme Gigabit Ethernet
Interface Type	Ethernet
Hardware Address	00-13-72-AD-8C-9A
Connection Name	Local Area Connection
Connection Speed	1000 Mbps
MTU	1500 bytes
IP / Subnet Mask	10.204.2.76 / 255.255.0.0
Gateway	10.204.1.250
DHCP	10.210.1.100
WINS	10.210.1.100
WINS	10.210.1.130
WINS	10.211.1.201
DNS	10.210.1.100
DNS	10.210.1.130
DNS	10.211.1.201

Login Performance

Comparing the active sessions on the web server (Figure 2), and the reported virtual user count from the load tool (Figure 3), it can be seen that the number of sessions are the same. There is a slight offset, but this can be explained by the time to login and report the active session to Perfmon.

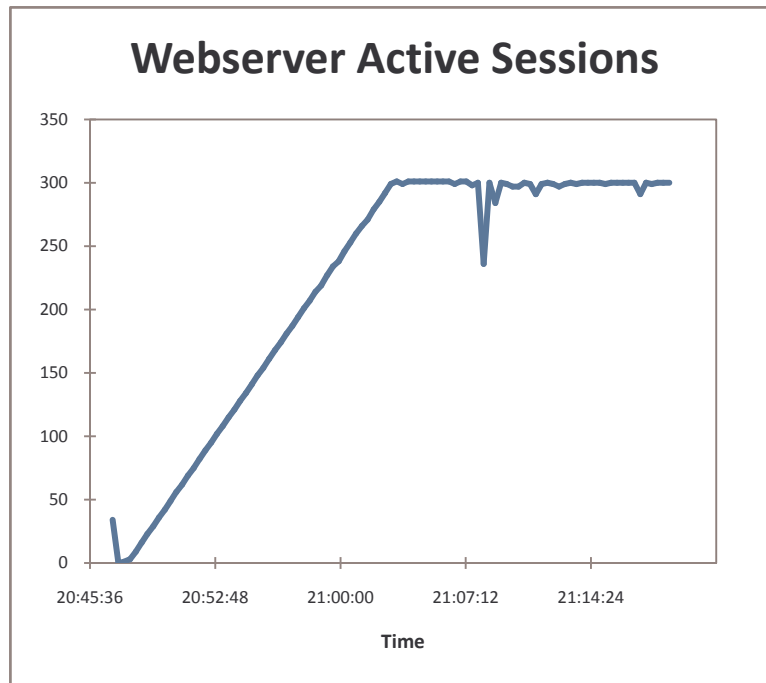


Figure 2 - Login Web Server Session

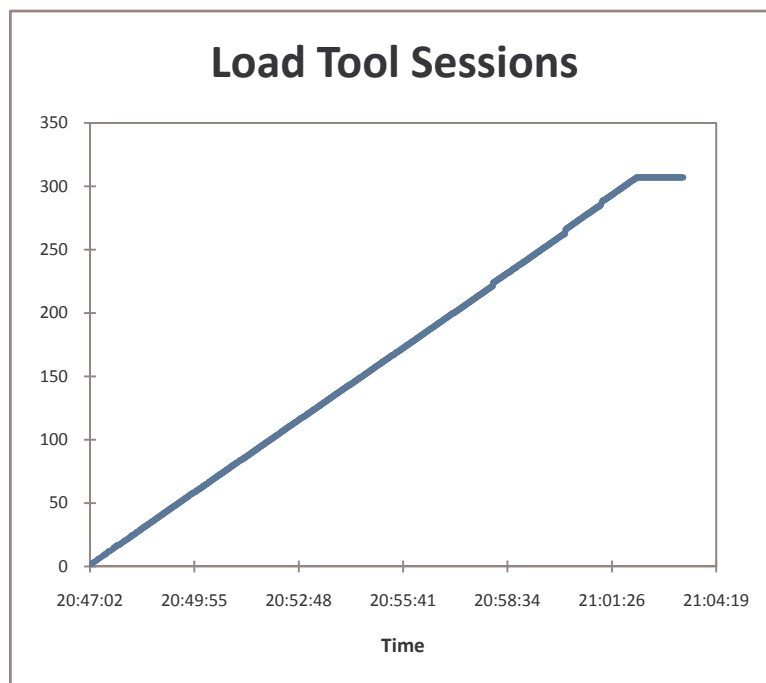


Figure 3 - Login Load Tool Reported Sessions

In Figure 4 below, the number of actions per second cease to grow around 20:56:00, even though the number of users (and load) continues to grow. This indicates a resource limitation.

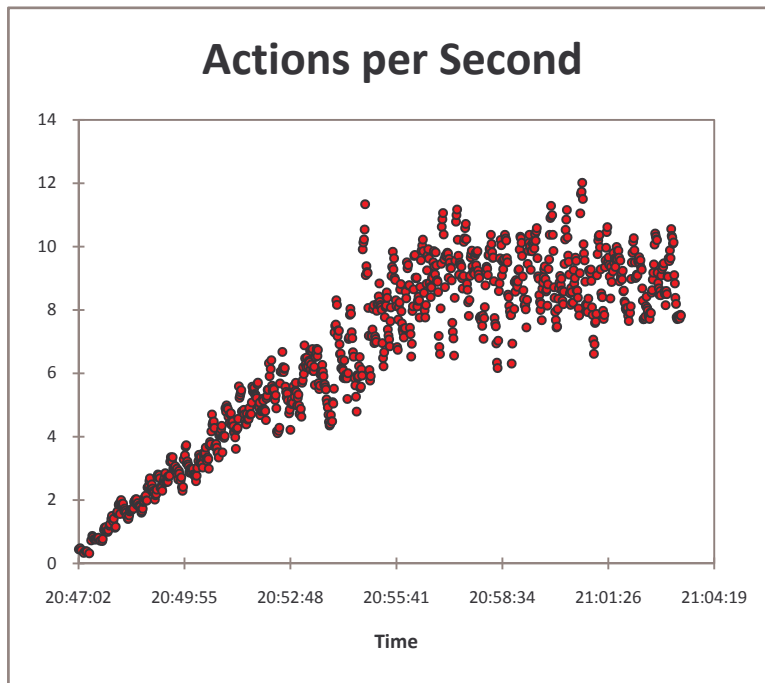


Figure 4 - Login Actions Attempted per Second

Viewing both Figure 5 and Figure 6, there is not much variation of interest. Although both do tend to gradually increase, there is no remarkable 'break' as can be seen in some other scenarios. Action Response Time does tend to show an upward turn around 20:55:41, but it is difficult to discern due to the wide variation of data.

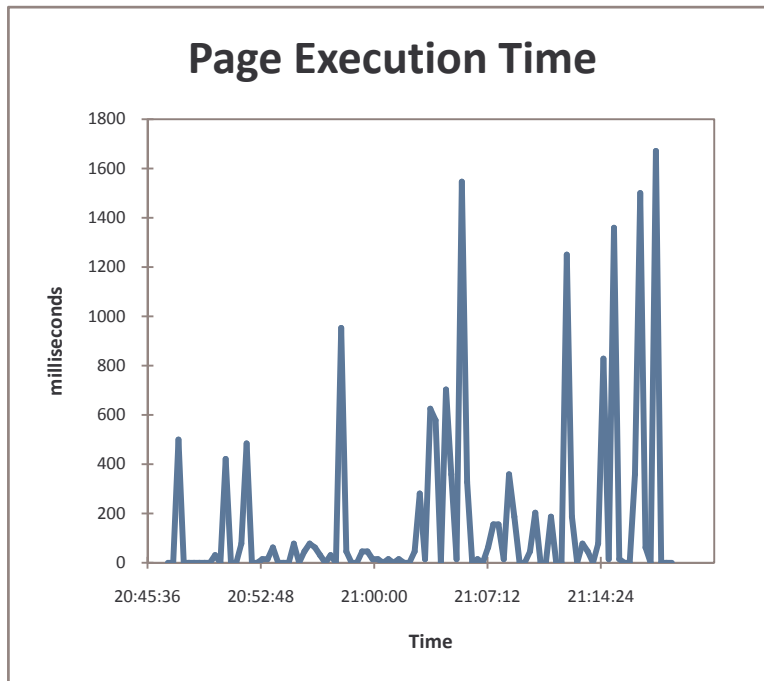


Figure 5 - Login Web Server Page Execution Time

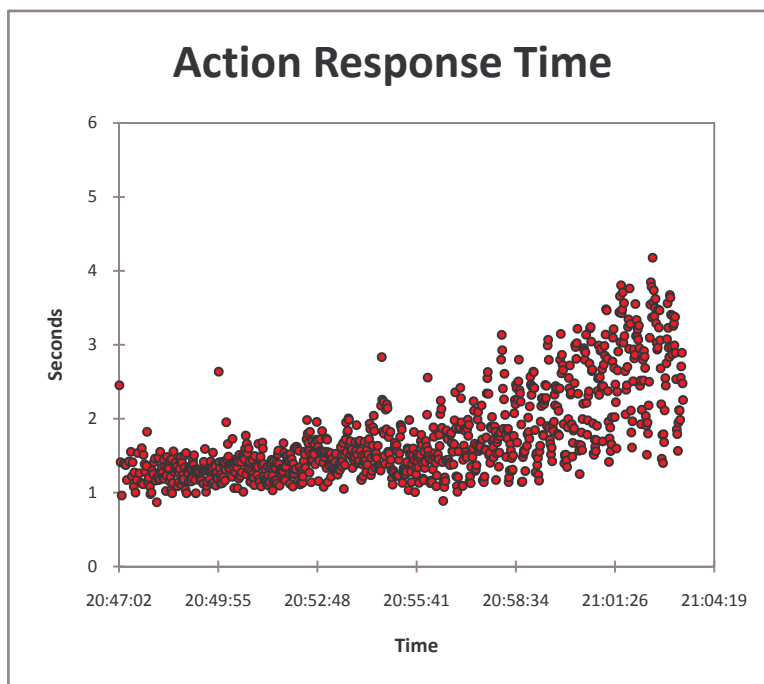


Figure 6 - Login Load Tool Reported Response Time

The memory data in Figure 7, Figure 8, and Figure 9 show a steady variation that is attributable to the load level shown in Figure 2.

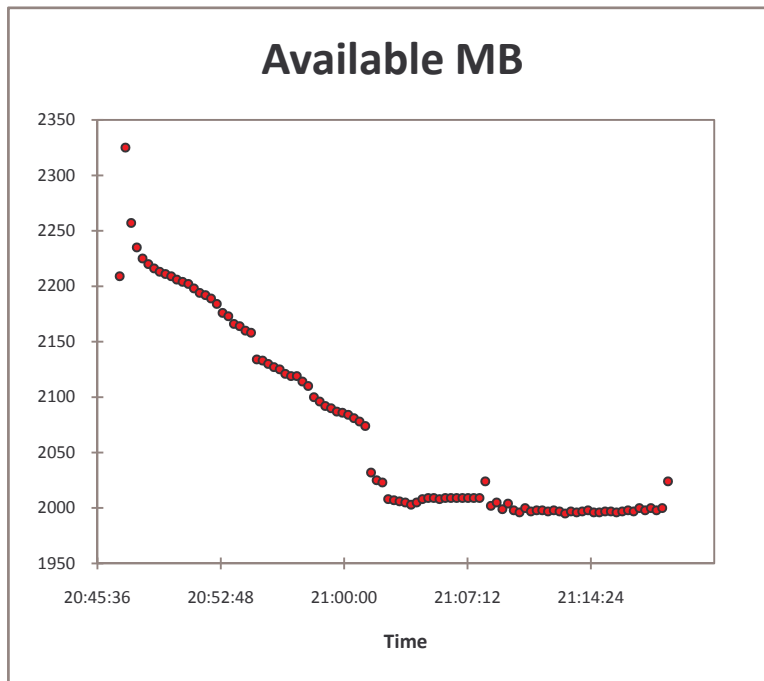


Figure 7 - Login Web Server Available Memory

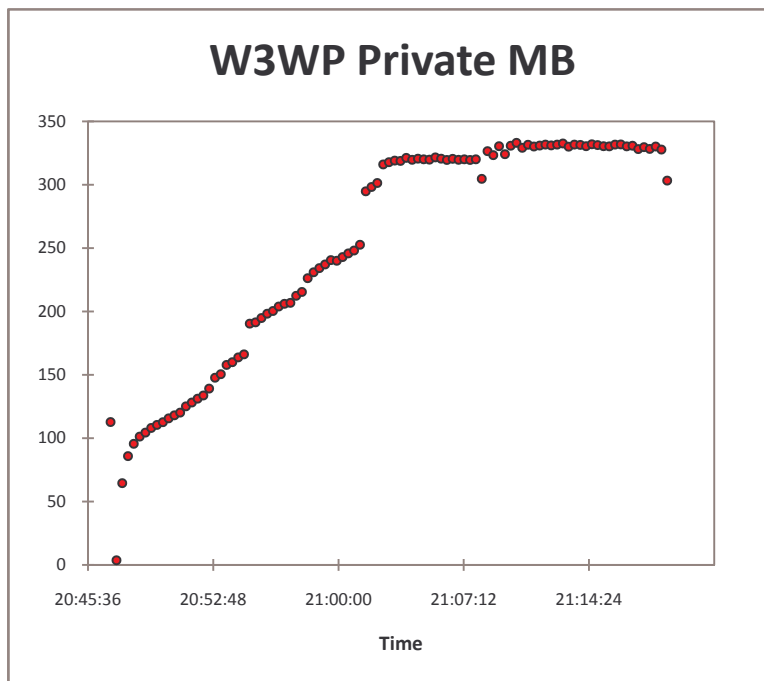


Figure 8 - Login Worker Process Private Memory

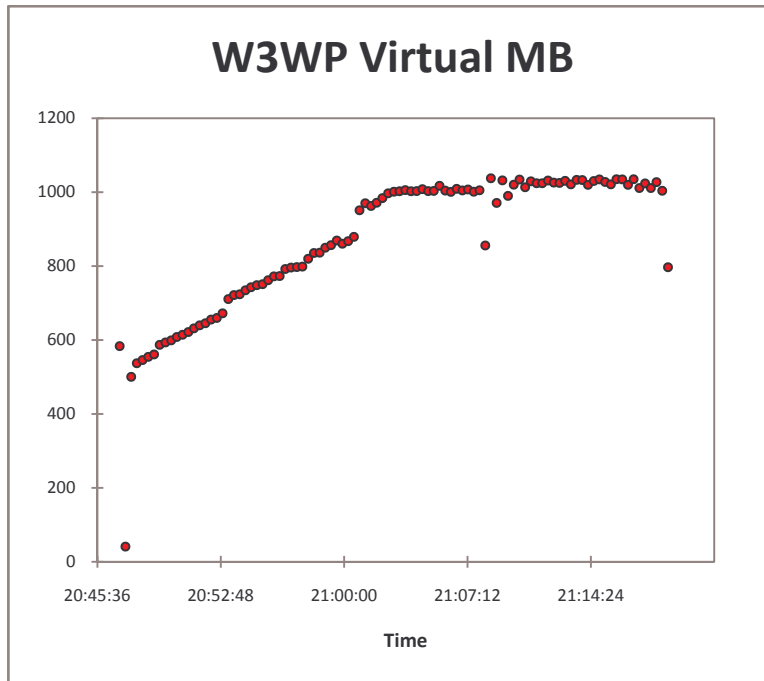


Figure 9 - Login Worker Process Virtual Memory

The CPU utilization shown in Figure 10 does not show any remarkable breaks or turns. The leveling around 21:04:00 is attributable to the leveling of load.

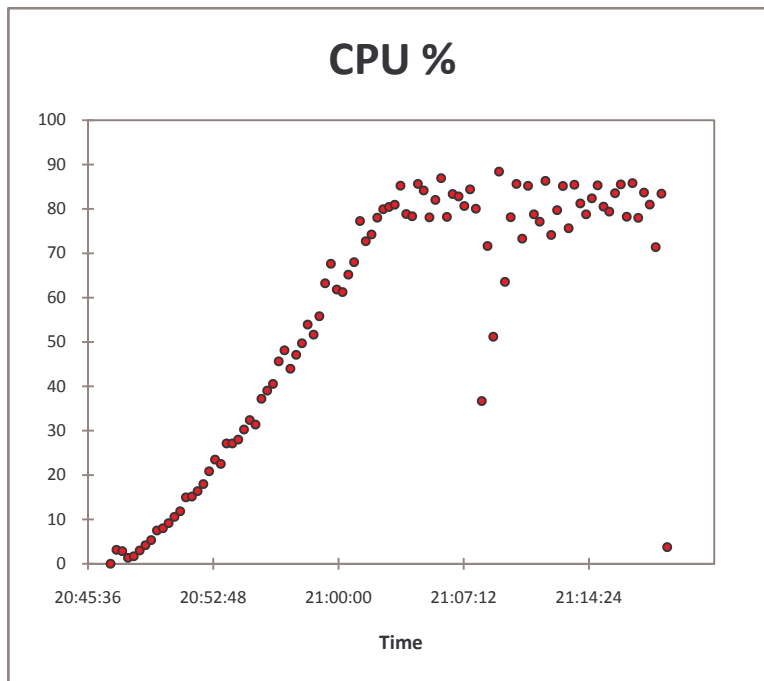


Figure 10 - Login Processor Utilization

Both context switching in Figure 11 and queueing in Figure 12 show a marked upturn in thread handling around the test time of 20:55:00, which is in line with the leveling of action rate before.

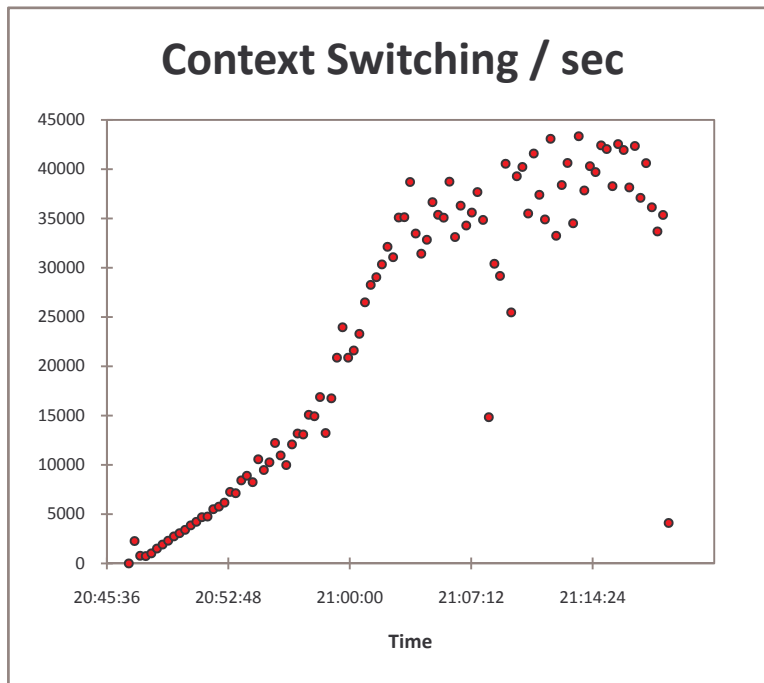


Figure 11 - Login Context Switching

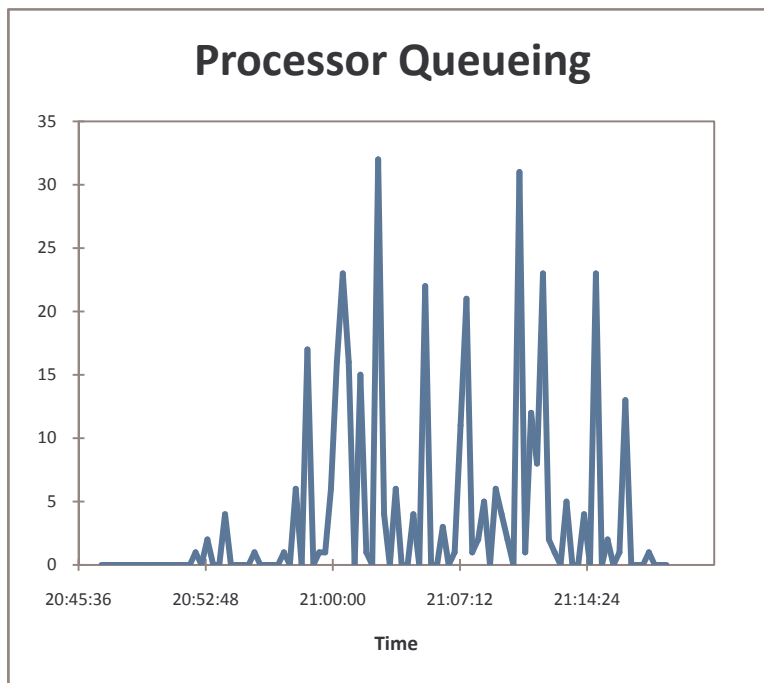


Figure 12 - Login Processor Queueing

Analysis

Given the correlation between the leveling of action rate, and the increase in context switching and processor queueing, both at the test time of 20:55:00, the resource of limitation in logins in this environment is the number of CPU cores.

Taking an average of the request rate at the breakage point, it is determined that the rate at which logins saturate the server is 8.06 actions per second.

COLD Rendering

Raw Data

Comparing the active sessions on the web server (Figure 13), and the reported virtual user count from the load tool (Figure 14), it can be seen that the number of sessions are the same. There is a slight offset, but this can be explained by the time to login and report the active session to Perfmon.

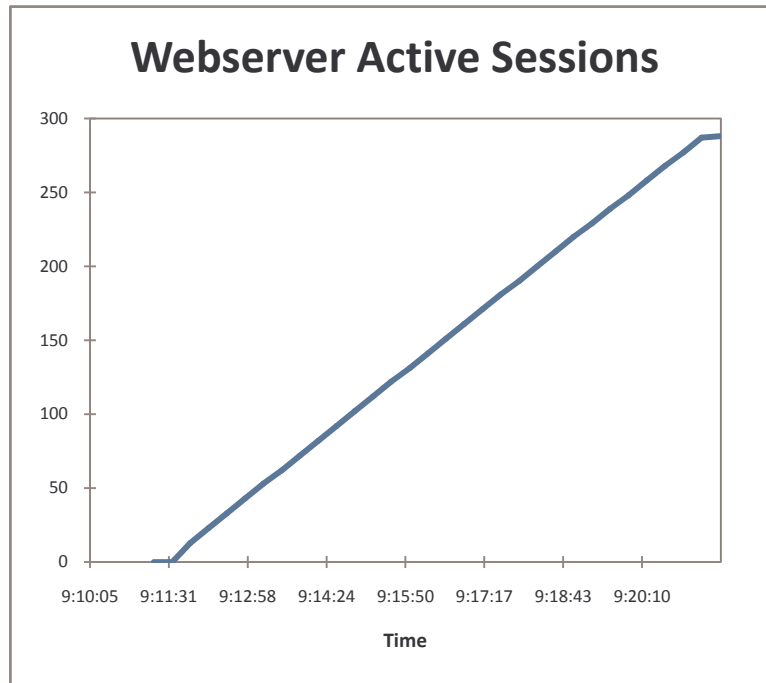


Figure 13 – COLD Web Server Sessions

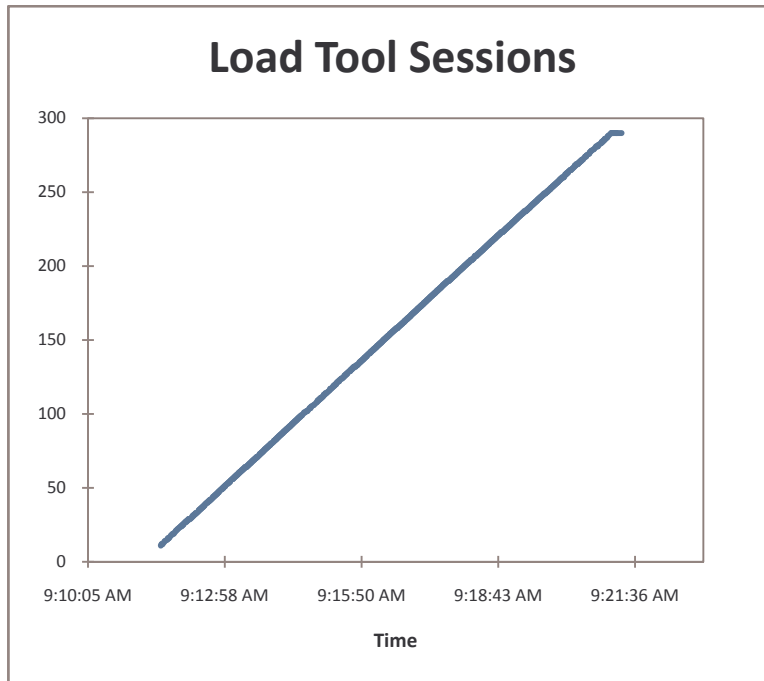


Figure 14 – COLD Reported Load Tool Sessions

The basis for the sizing measurement is the Actions per Second achieved. This can be seen in Figure 15 for the COLD Retrieval. In the next few figures, the limiting point of the server can be observed.

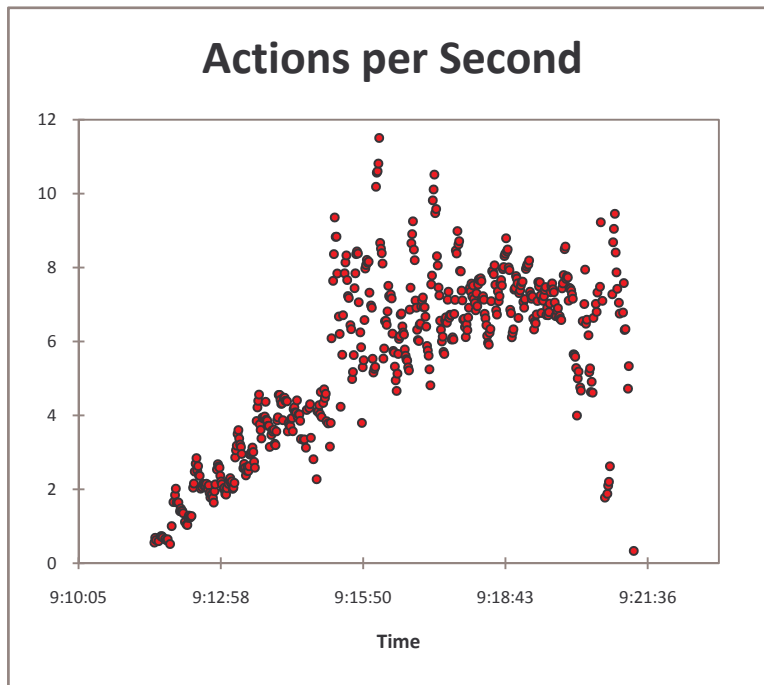


Figure 15 – COLD Actions per Second

In the following graphs, both Figure 16 and Figure 17, a breaking point is easily seen. At 09:20:00, both Execution Time and the Response time reported from the load tool. This indicates that a constraint was reached on the web server, and that maximum load that the server can handle was reached at this point.

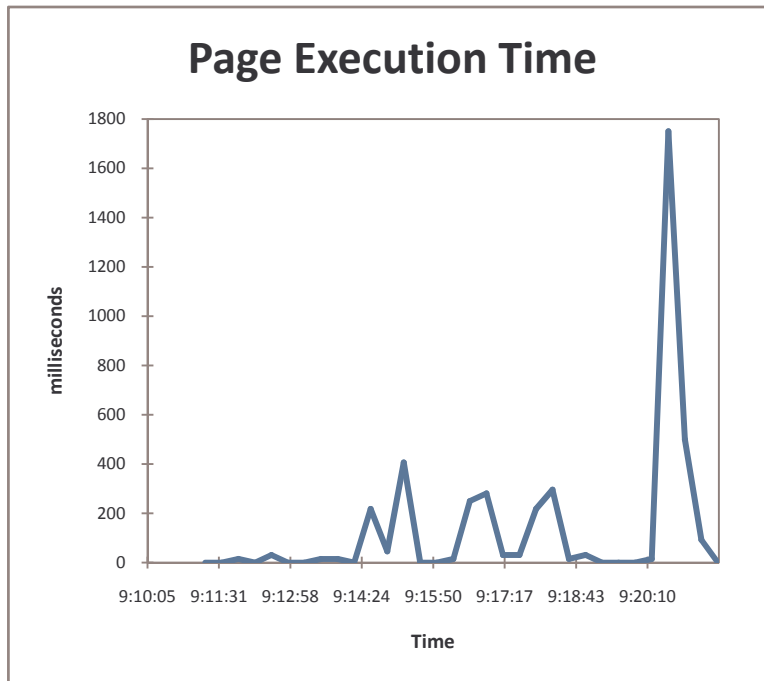


Figure 16 – COLD Page Execution Time

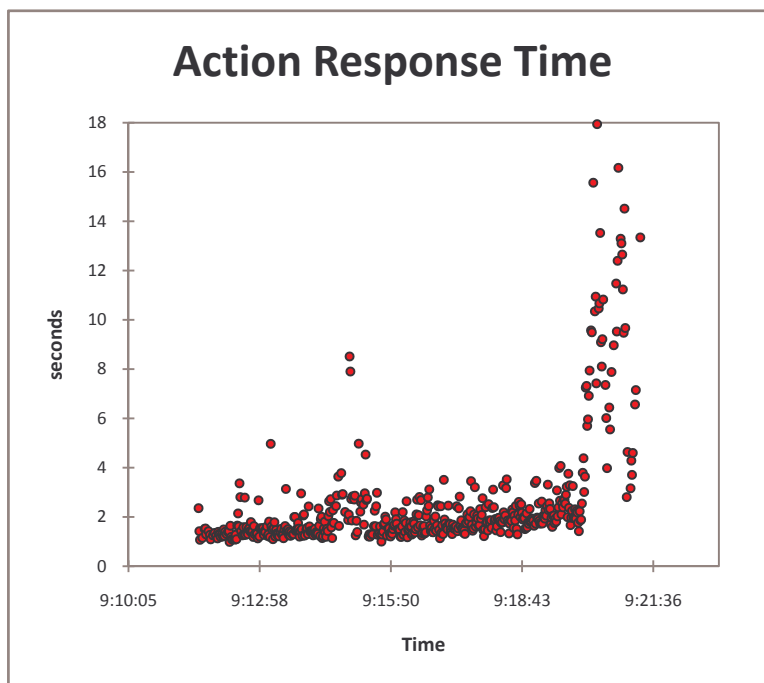


Figure 17 – COLD Action Response Times

The memory during the COLD test followed the number of virtual users in a linear trend. Both Figure 18 , Figure 19, and Figure 20 show this relationship.

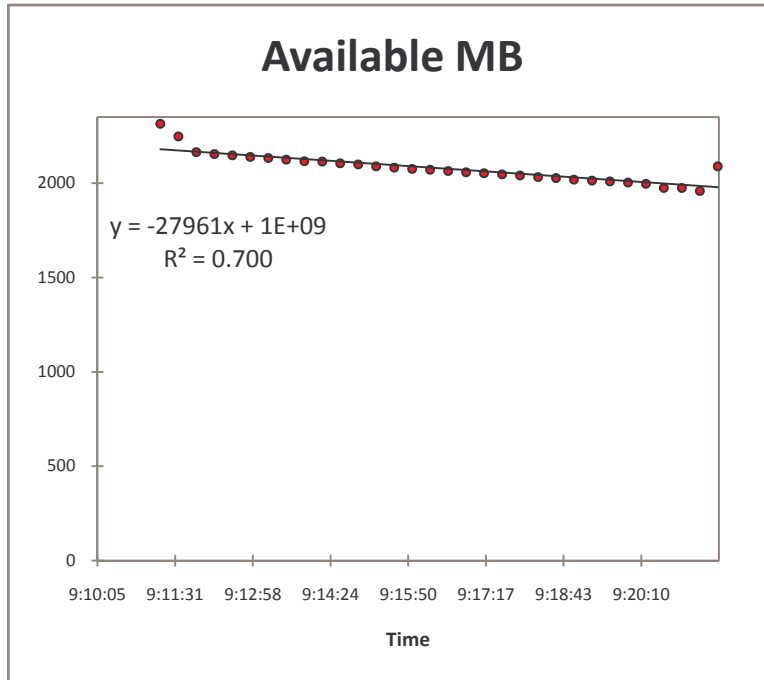


Figure 18 – COLD Available Memory

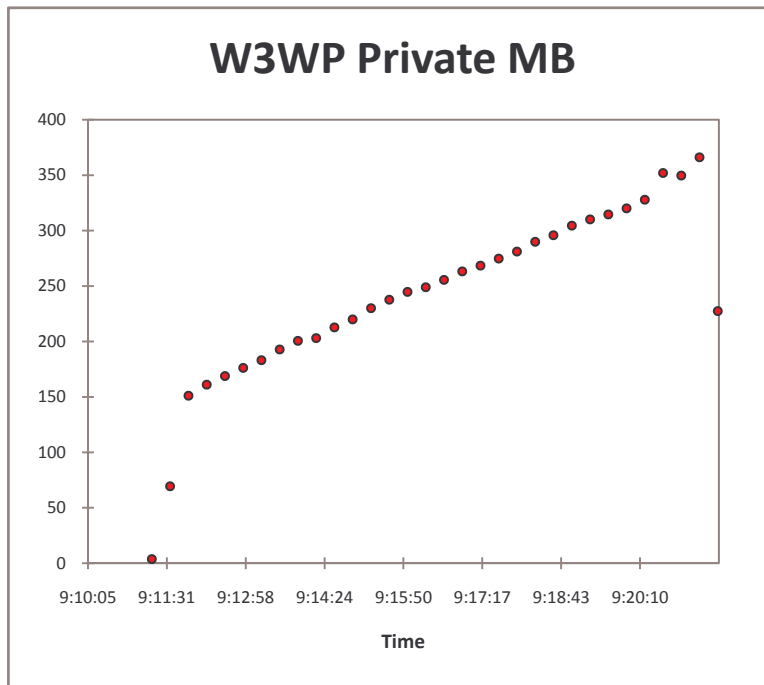


Figure 19 – COLD Worker Process Private Bytes

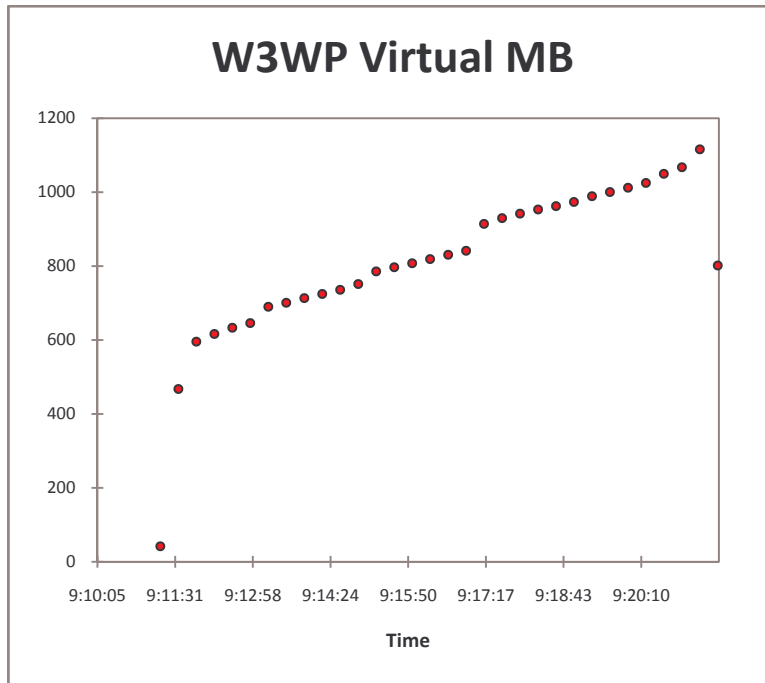


Figure 20 – COLD Worker Process Virtual Bytes

CPU utilization shows the source of the limitation on the web server. Figure 21 shows that the full processing power is never used. Looking at Figure 22 and Figure 23, the source of the limitation is seen to be the switching and queueing between threads.

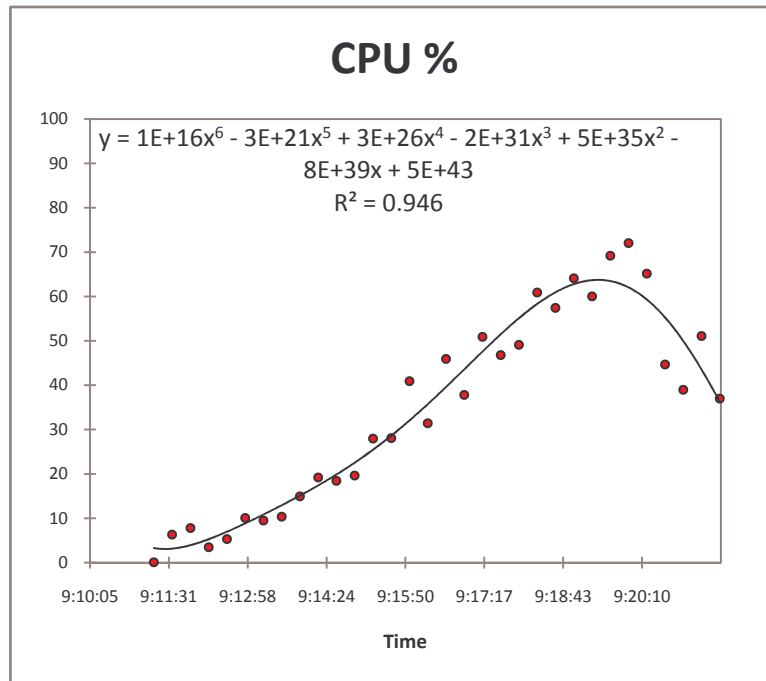


Figure 21 – COLD CPU Utilization

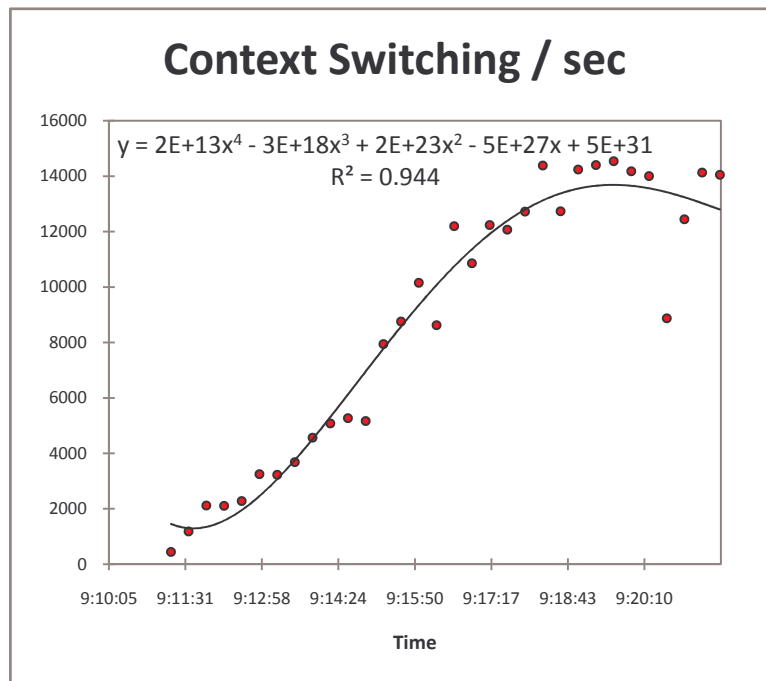


Figure 22 – COLD CPU Context Switching

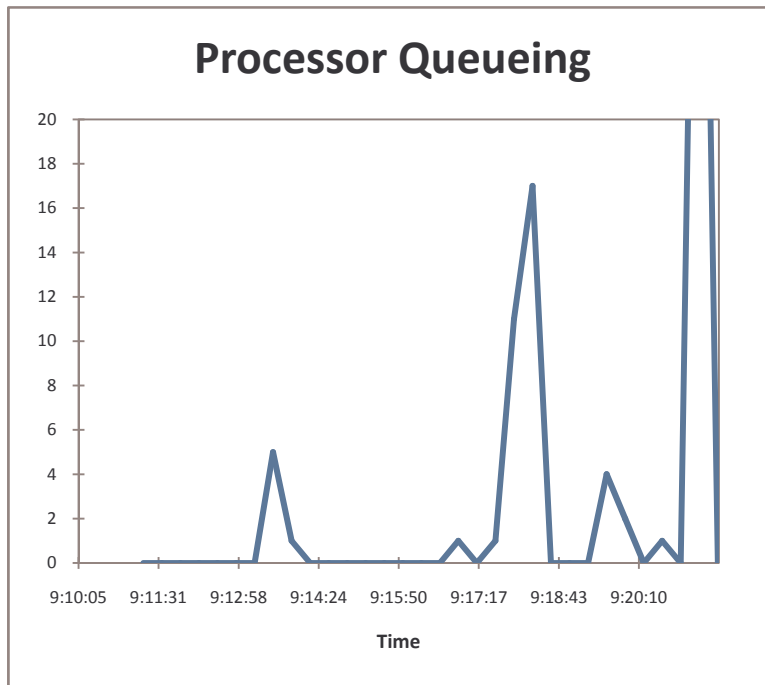


Figure 23 – COLD Processor Queueing

Analysis

Given the high response time and page execution, coupled with high context switching and processor queueing, the break can be located at a test time of 9:20:00. At this point, the statistic of interest is the average request rate, which is calculated to be 7.25 requests per second. This value will be used in final analysis.

Image Rendering

Raw Data

In the image tests, the load tool database logging stopped logging halfway through the test, but other logging continued. As seen below, the session number can be inferred from the web server and load tool session graphs in Figure 24 and Figure 25.

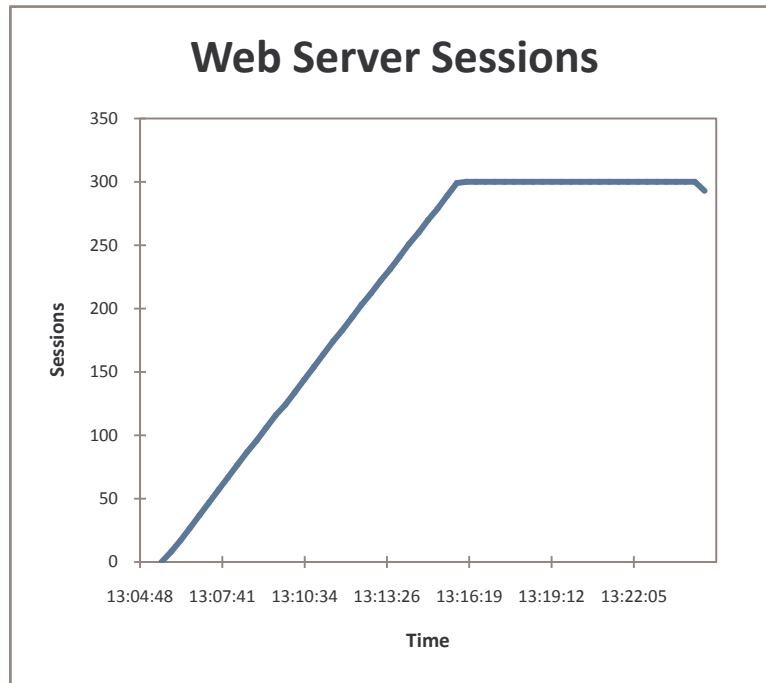


Figure 24 – Image Test Web Server Sessions

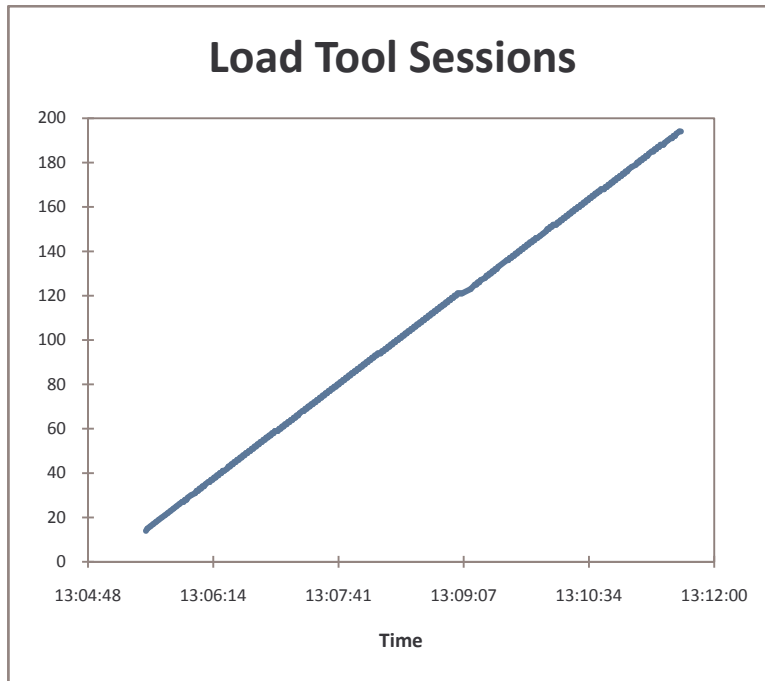


Figure 25 – Image Test Load Tool Sessions

The actions per second in Figure 26, although incomplete, shows a steady growth in the actions per second, allowing for a predictive measurement for further computation.

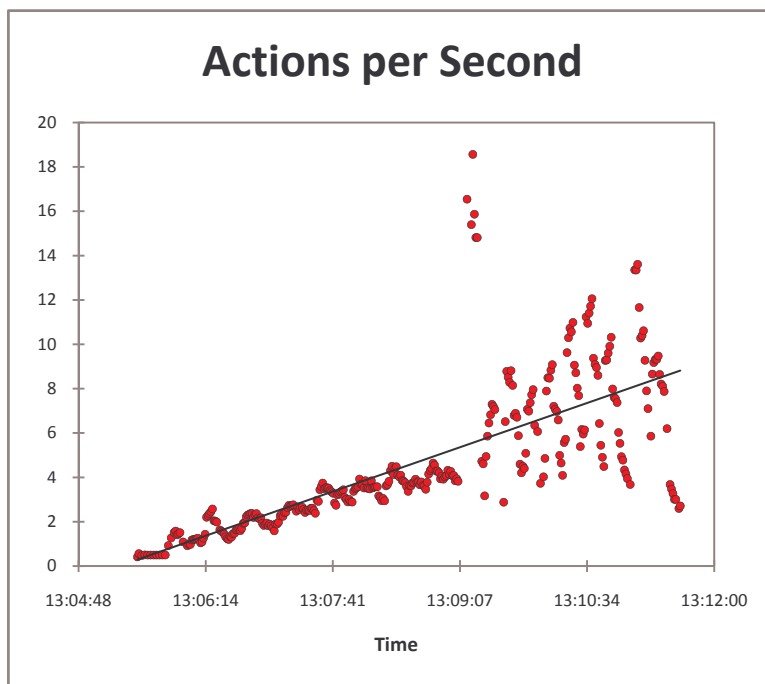


Figure 26 – Image Test Load Tool Reported Actions per Second

Combining the focus on both Figure 27 and Figure 28, the spike at 13:07:41 of execution time can be ignored, due to no effect on the actual response time client side. However, there are spikes on both graphs around 13:11:00.

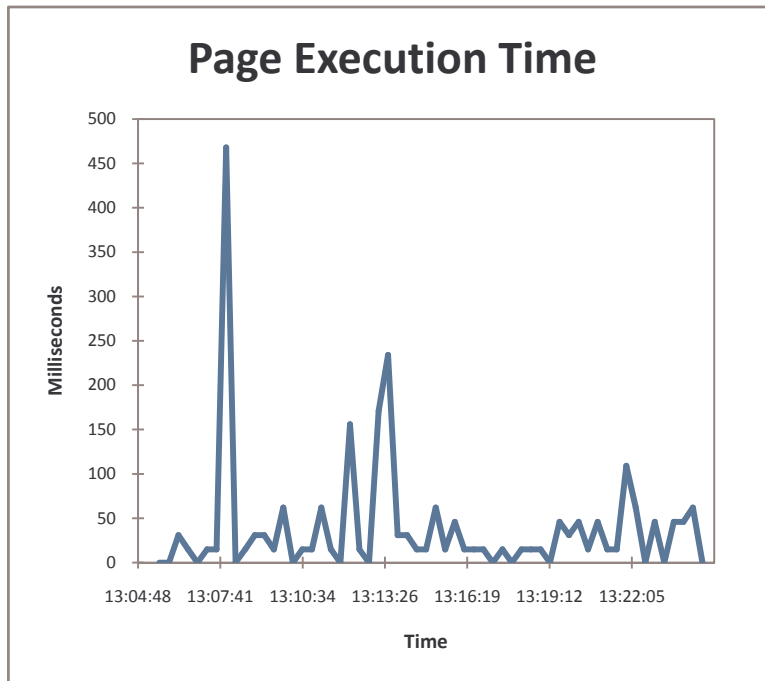


Figure 27 – Image Test Web Page Execution Time

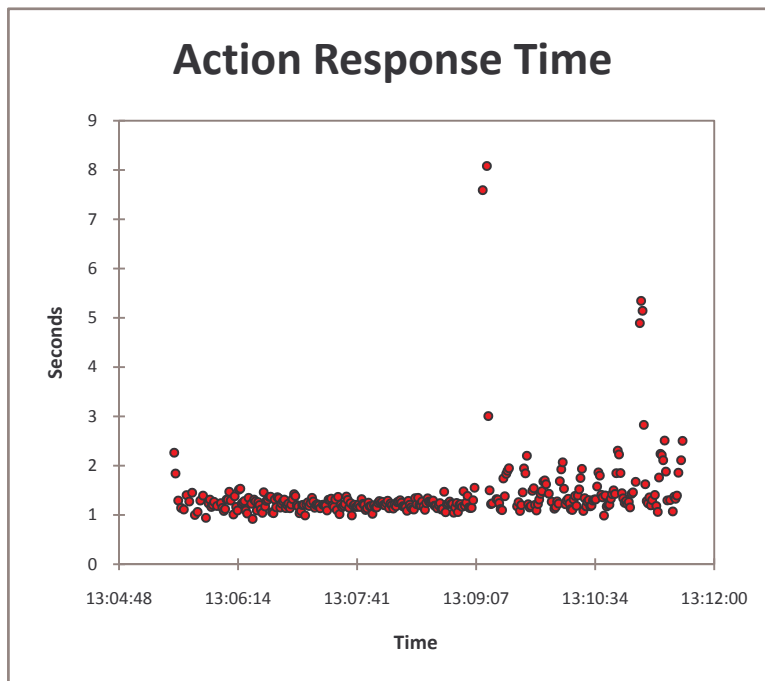


Figure 28 – Image Test Load Tool Action Response Time

The memory during the COLD test followed the number of virtual users in a linear trend. Figure 29, Figure 30, and Figure 31 show this relationship.

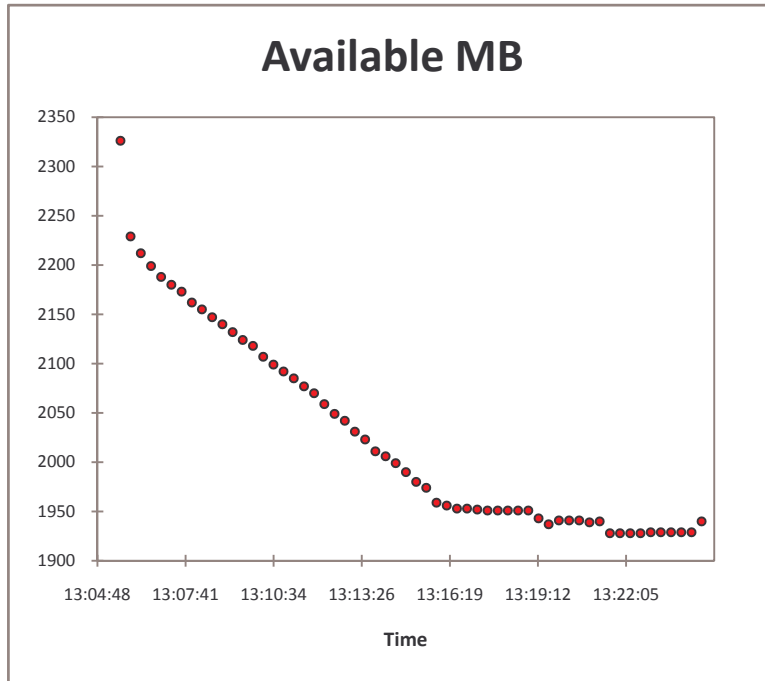


Figure 29 – Image Test Web Server Available Memory

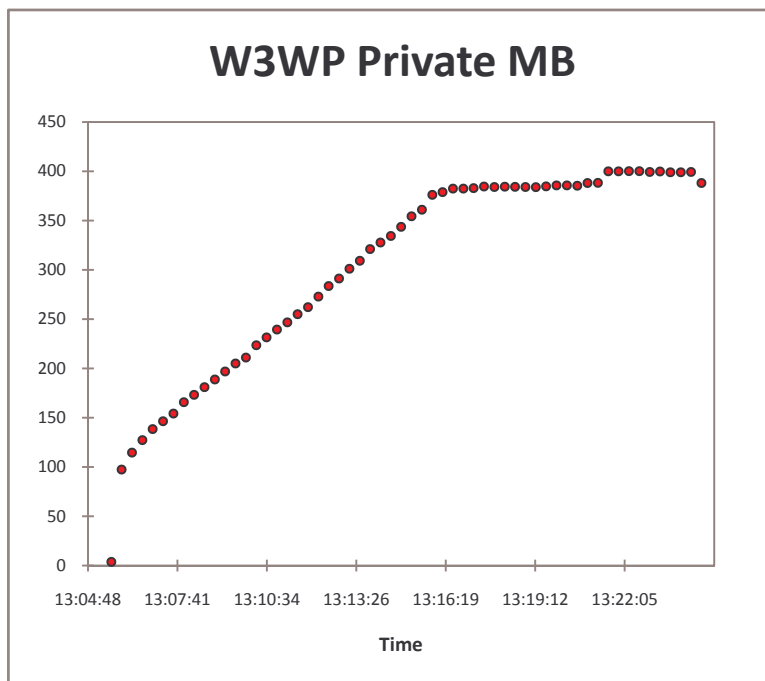


Figure 30 – Image Test Worker Process Private Bytes

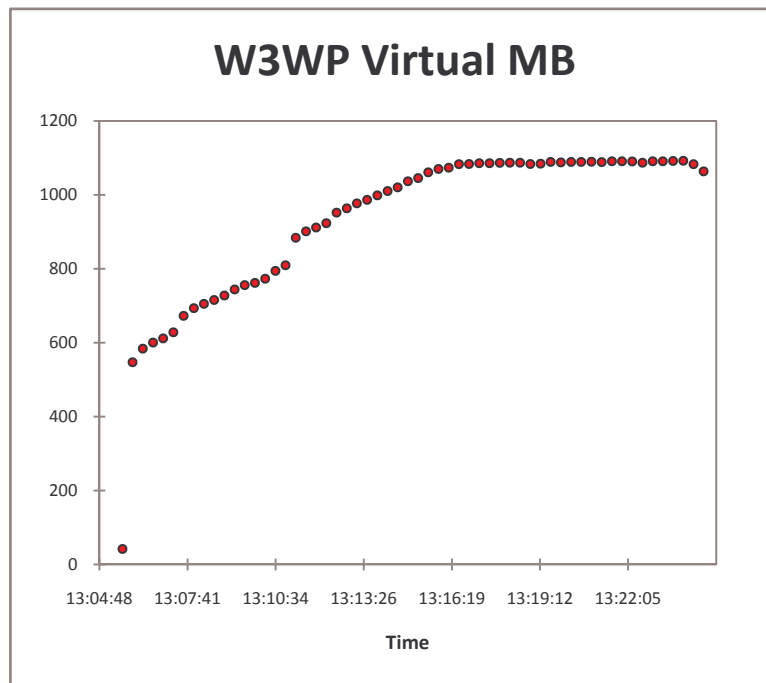


Figure 31 – Image Testing Worker Process Virtual Bytes

Again, Figure 32 shows that not all of the CPU available was used at full load.

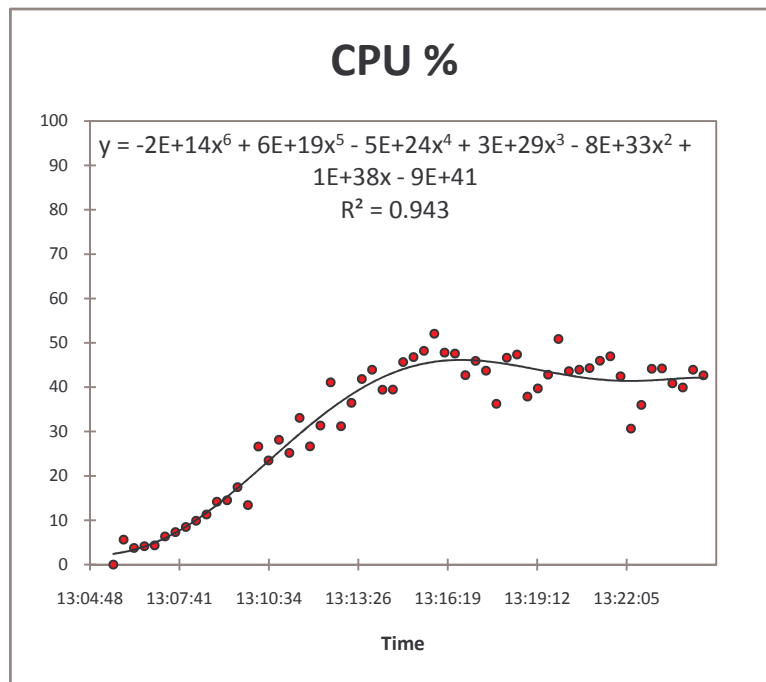


Figure 32 – Image Test CPU Utilization

Context Switching and Processor Queueing in Figure 33 and Figure 34 show that the limitation reached on the web server was that of switching between processing threads. The recommended value for processor queue is less than $2 * \# \text{ CPUs}$. As the web server is a hyper threaded dual processor box, the maximum value is 8.

Here, it is difficult to tell exactly when the server 'breaks'. With the other performance counters though, it can be seen that there is a break around 13:11:00.

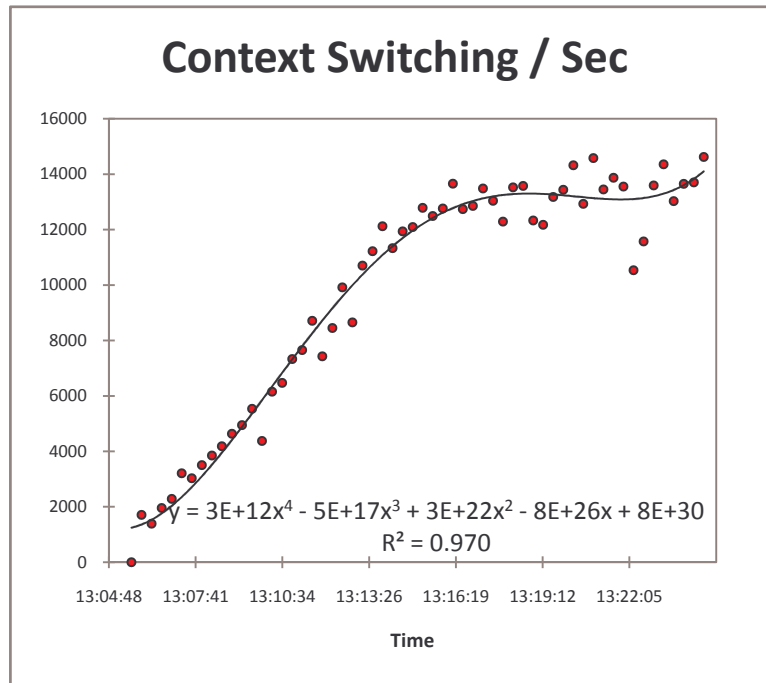


Figure 33 – Image Test Context Switching

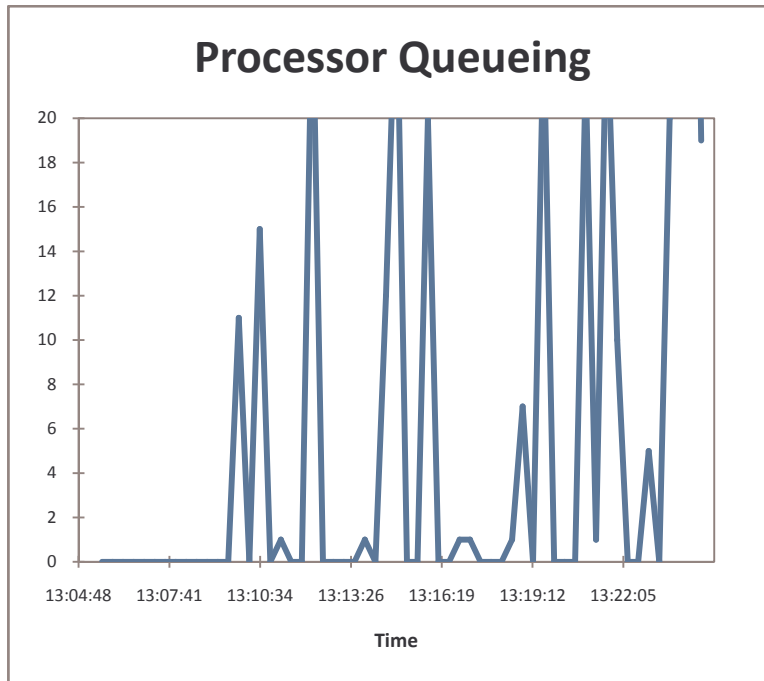


Figure 34 – Image Test Processor Queueing

Analysis

In both page execution time and response time, there is a spike around test time 13:11:00. In the processor queueing, even though there are moderate spikes earlier, the first major spike occurs at this point as well.

Taking an average, the value of interest is found to be 8.25 requests per second.

AFP Rendering

Raw Data

The web server and load tool session counts are shown to be matching in Figure 35 and Figure 36.

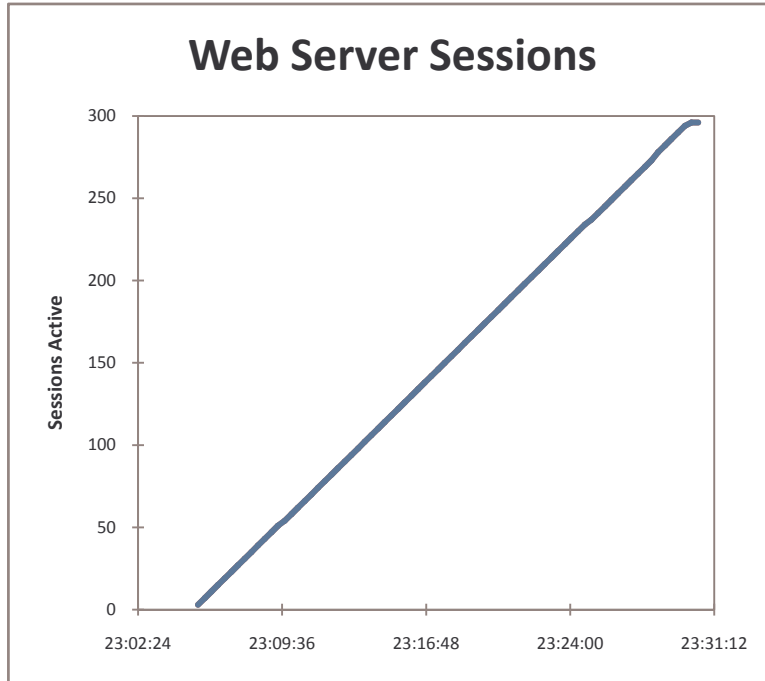


Figure 35 – AFP Web Server Sessions

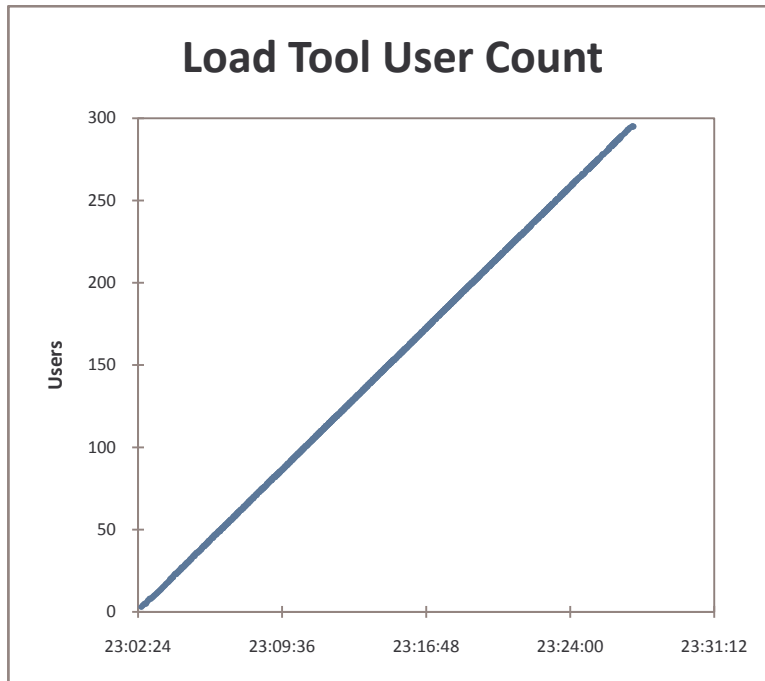


Figure 36 – AFP Load Tool Recorded Session Count

Both requested actions per second and page execution time (Figure 37 and Figure 38) show a break around test time 23:23:00. The Action Response Time (Figure 39) also shows an increase in time around the same time.

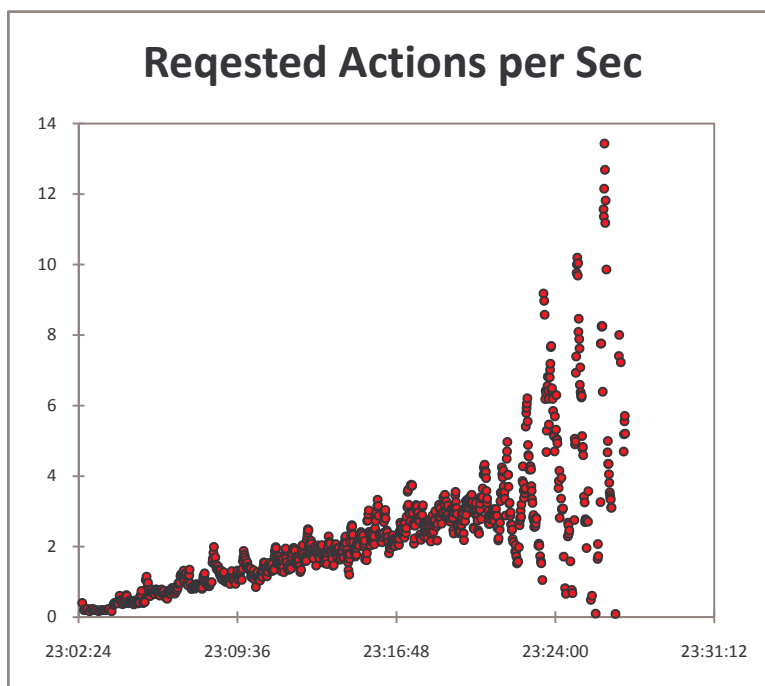


Figure 37 – AFP Requested Actions per Second

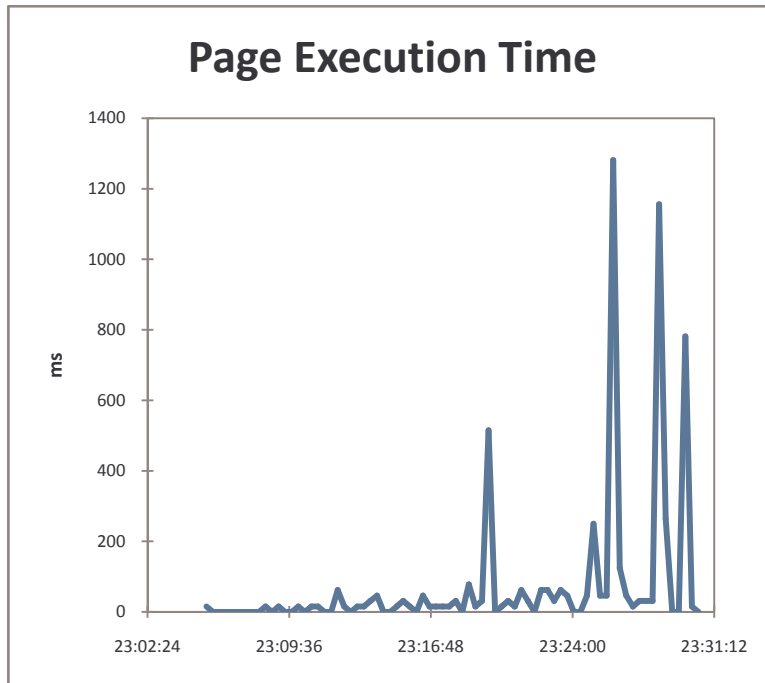


Figure 38 – AFP Page Execution Time

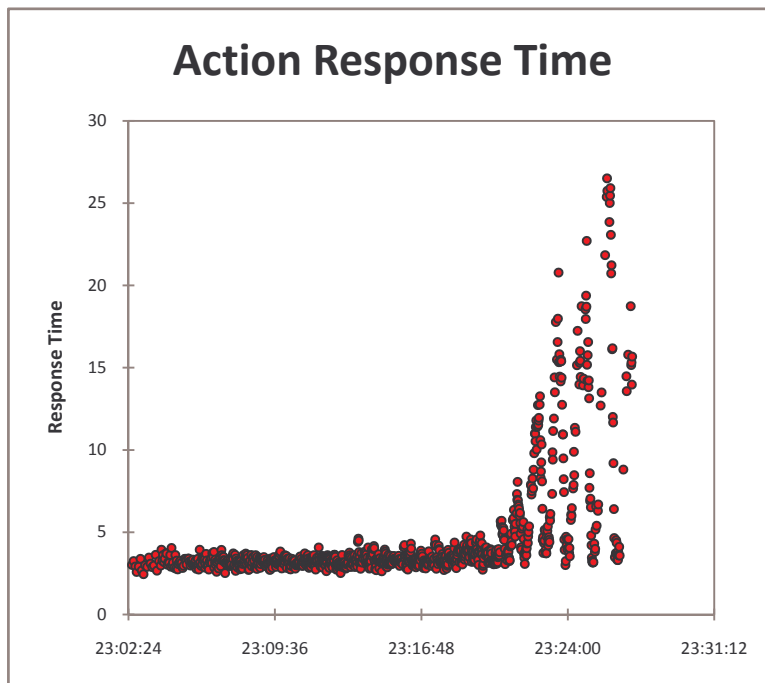


Figure 39 – AFP Load Tool Recorded Response Time

All three memory profiles (Figure 40, Figure 41 and Figure 42) show memory usage consistent with the user load.

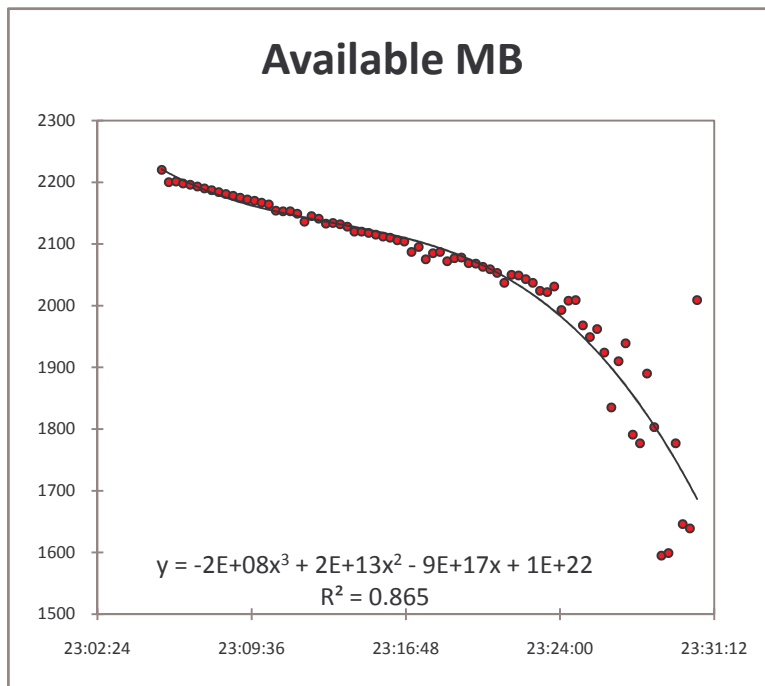


Figure 40 – AFP Available Memory

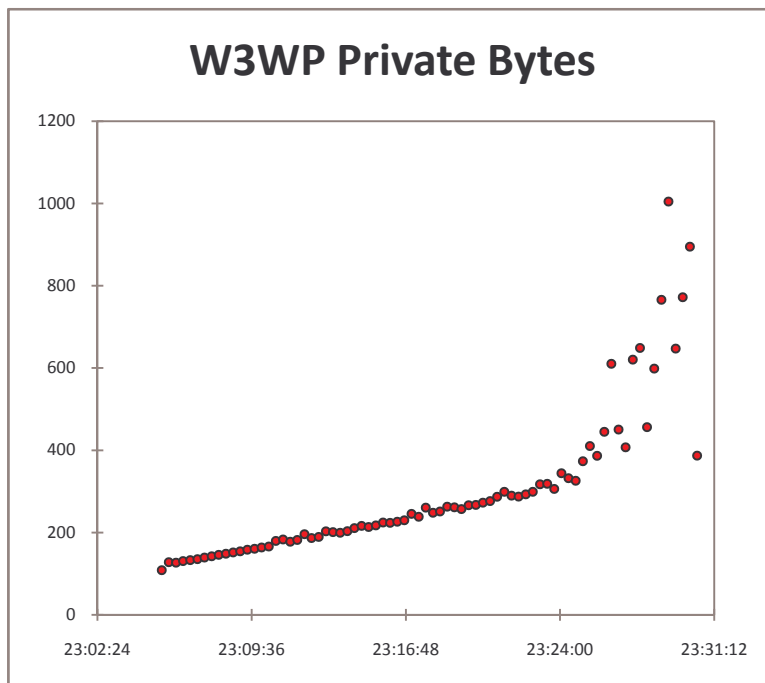


Figure 41 – AFP Worker Process Private Memory

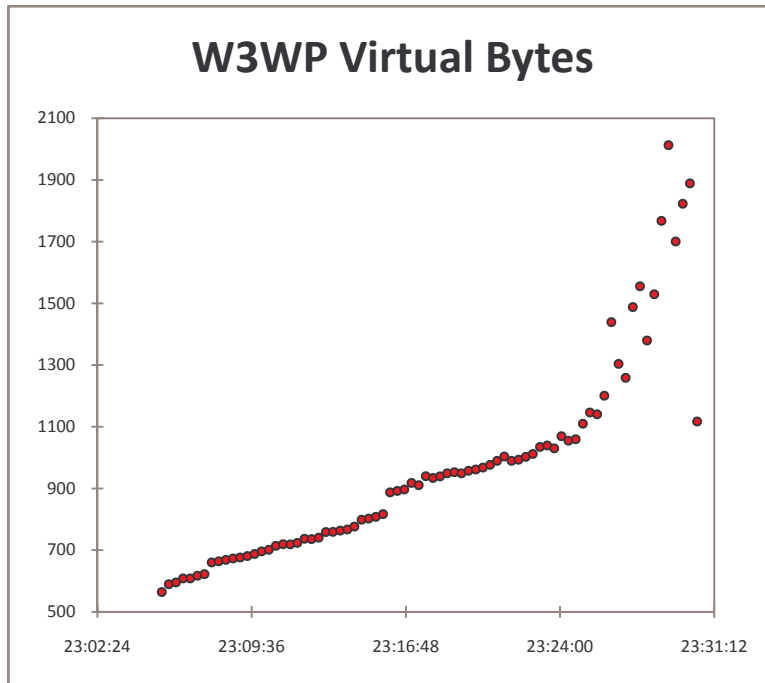


Figure 42 – AFP Worker Process Virtual Memory

Both Context Switching (Figure 44) and Processor Queueing (Figure 45) both show dramatic increases around the test time of 23:23:00. The CPU utilization (Figure 43) also goes over the recommended sustained value of 70% at this time.

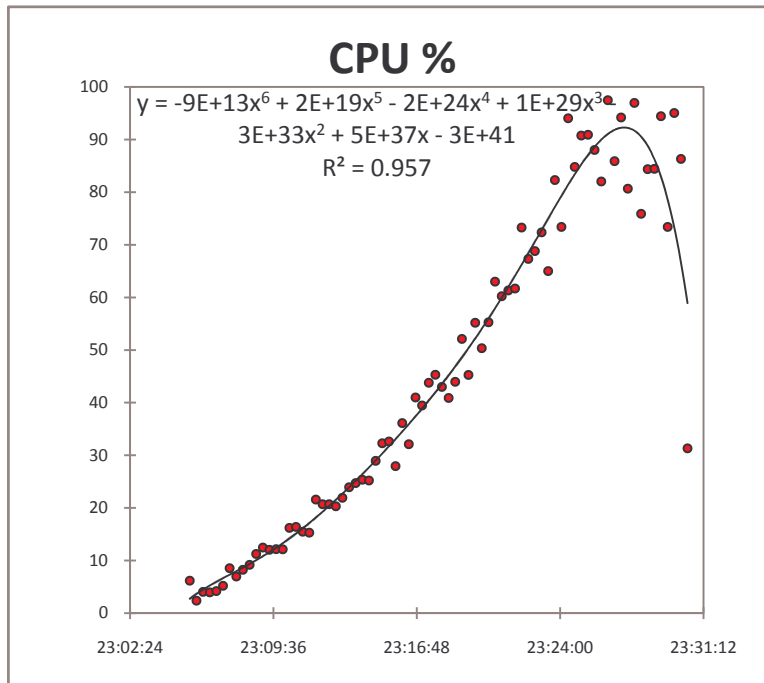


Figure 43 – AFP Processor Utilization

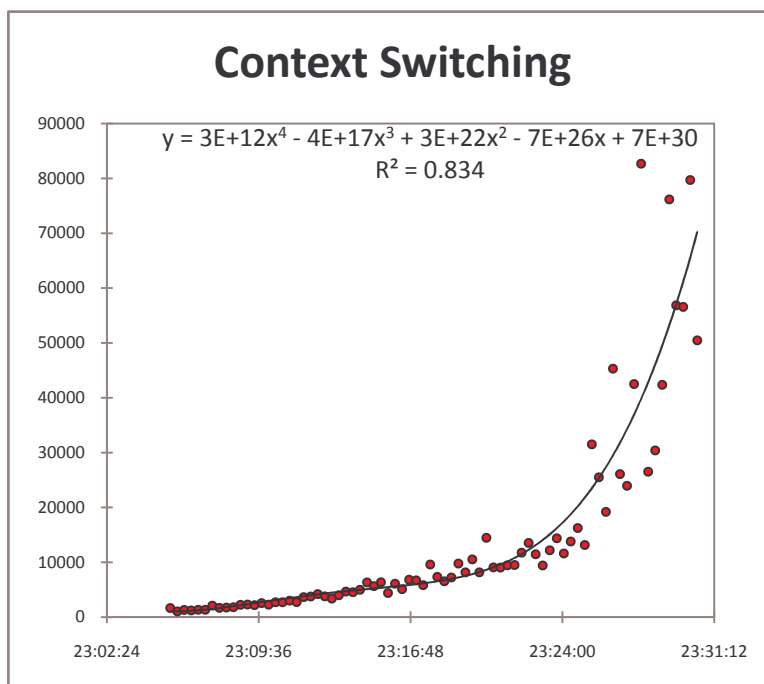


Figure 44 – AFP Processor Context Switching

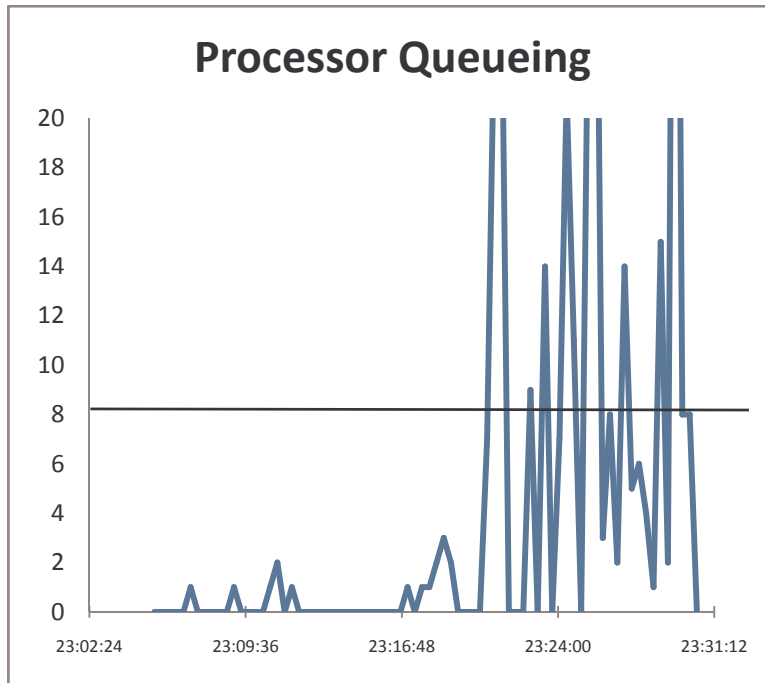


Figure 45 – AFP Processor Queueing

Analysis

All of the performance counters point towards the resource limitation of the server occurring during test time 23:23:00. At this point, Context Switching and Processor Queueing both go over recommended values.

Taking an average, the requests per second at the critical point is 2.79.

Bandwidth Effects

When analyzing the bandwidth effects on certain actions, a singleton (single user) trace of network activity was taken in order to analyze the effects of both network speed and network latency. The following analysis is for the single user action of viewing certain document types (Text, Image and AFP).

In the figures below, there is a full sweep of network speeds from 56 kbps to 2056 kbps, where the effects of bandwidth are negligible. In addition, there are several series from 0ms of latency to 100ms.

For COLD (text) documents, there is a base time of 4 seconds in retrieving the document. The large jump in the graph occurs around 256 kbps, where bandwidth becomes the major issue in transferring the data to the client.

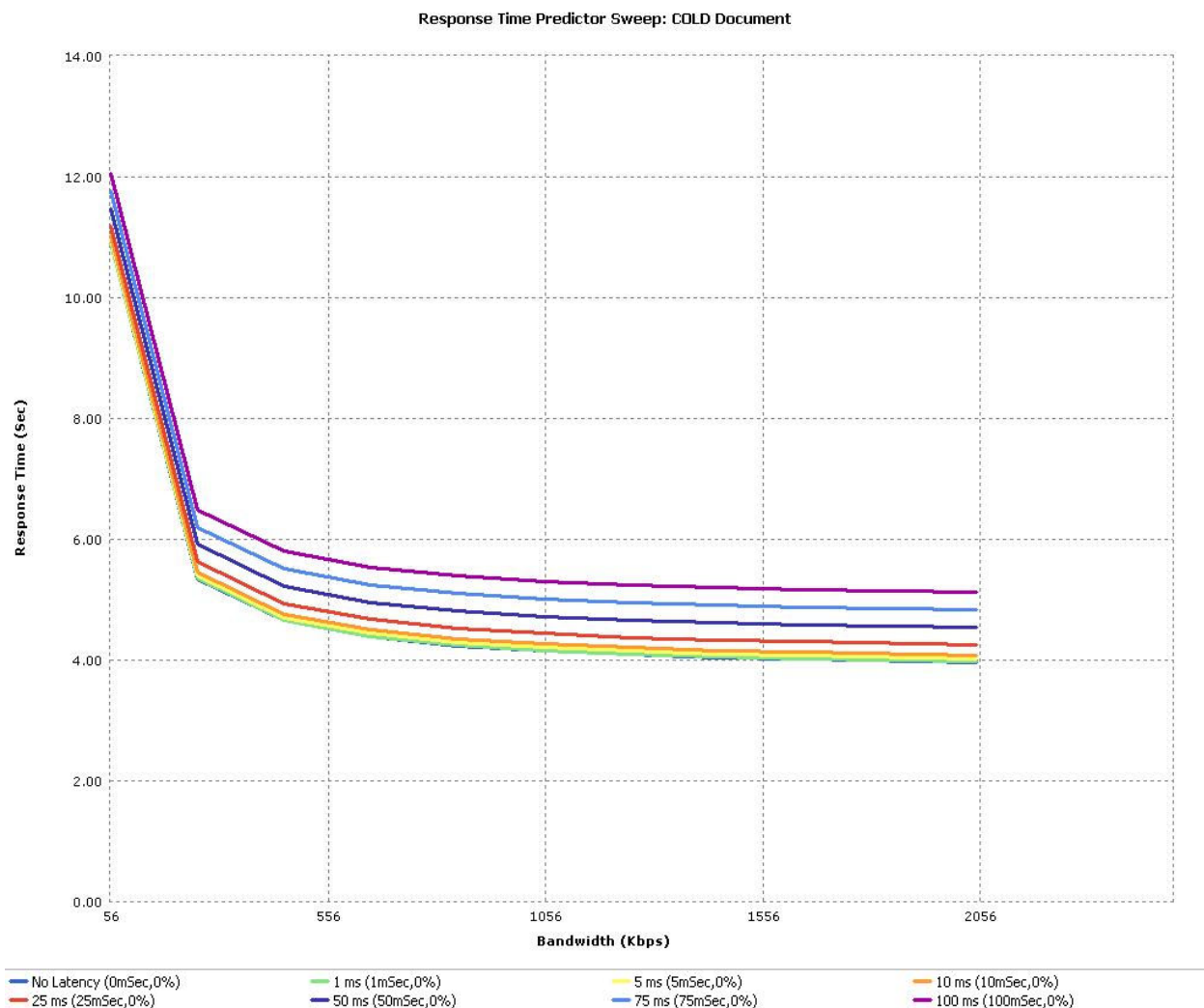


Figure 46 - Response Time Sweep of COLD Documents

A similar response at 256 kbps can be seen in the graph for image documents. In Figure 47, the minimum time can again be seen at 4 seconds, but with larger increases for latency over the previous text documents.

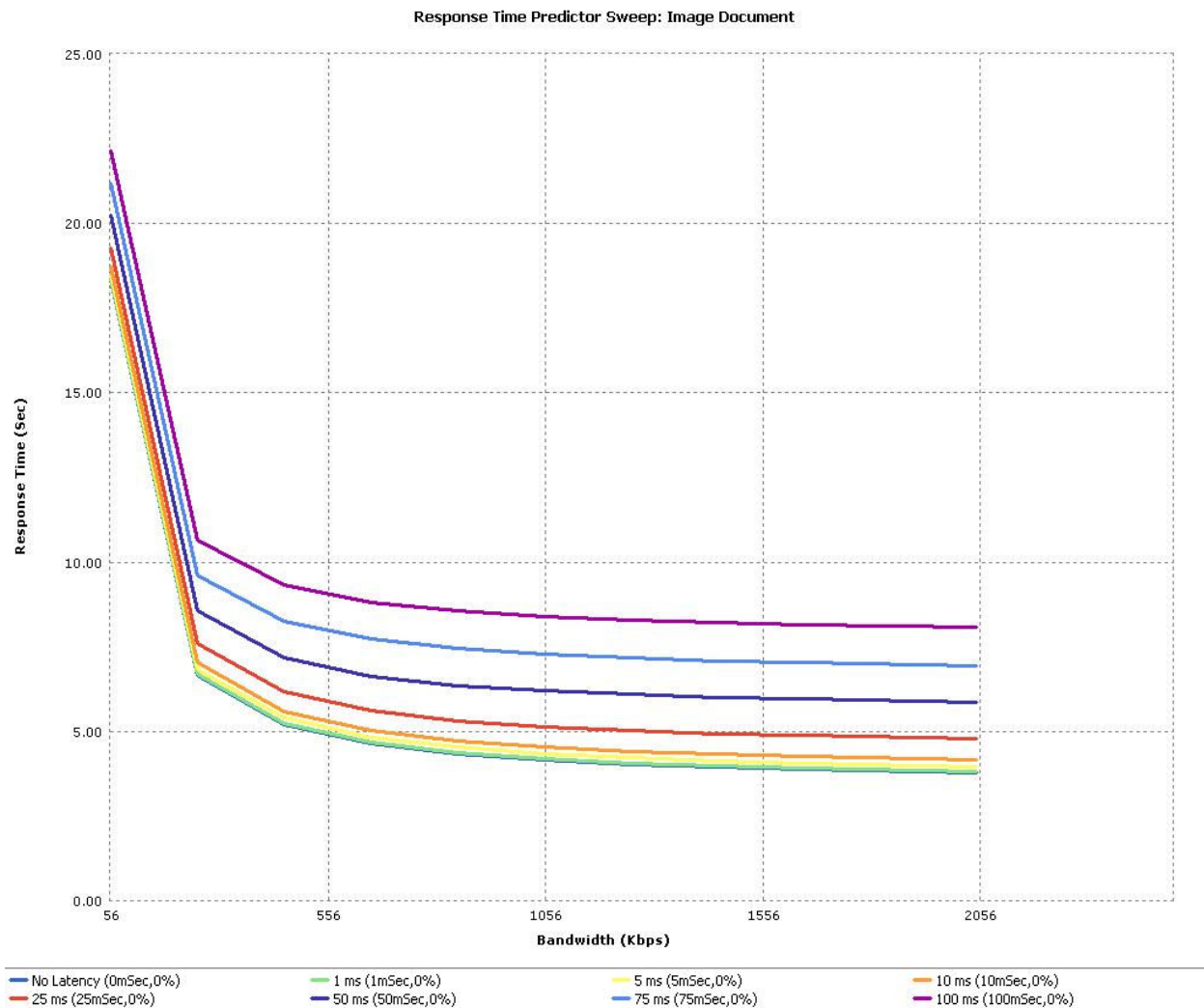


Figure 47 - Response Time Sweep of Image Documents

Again, for AFP documents, there is a sharp response at 256 kbps in the response time, shown in Figure 48. The minimum for this action is about 7.5 seconds, again with sharp increases for each level of latency past 10ms added to the network.

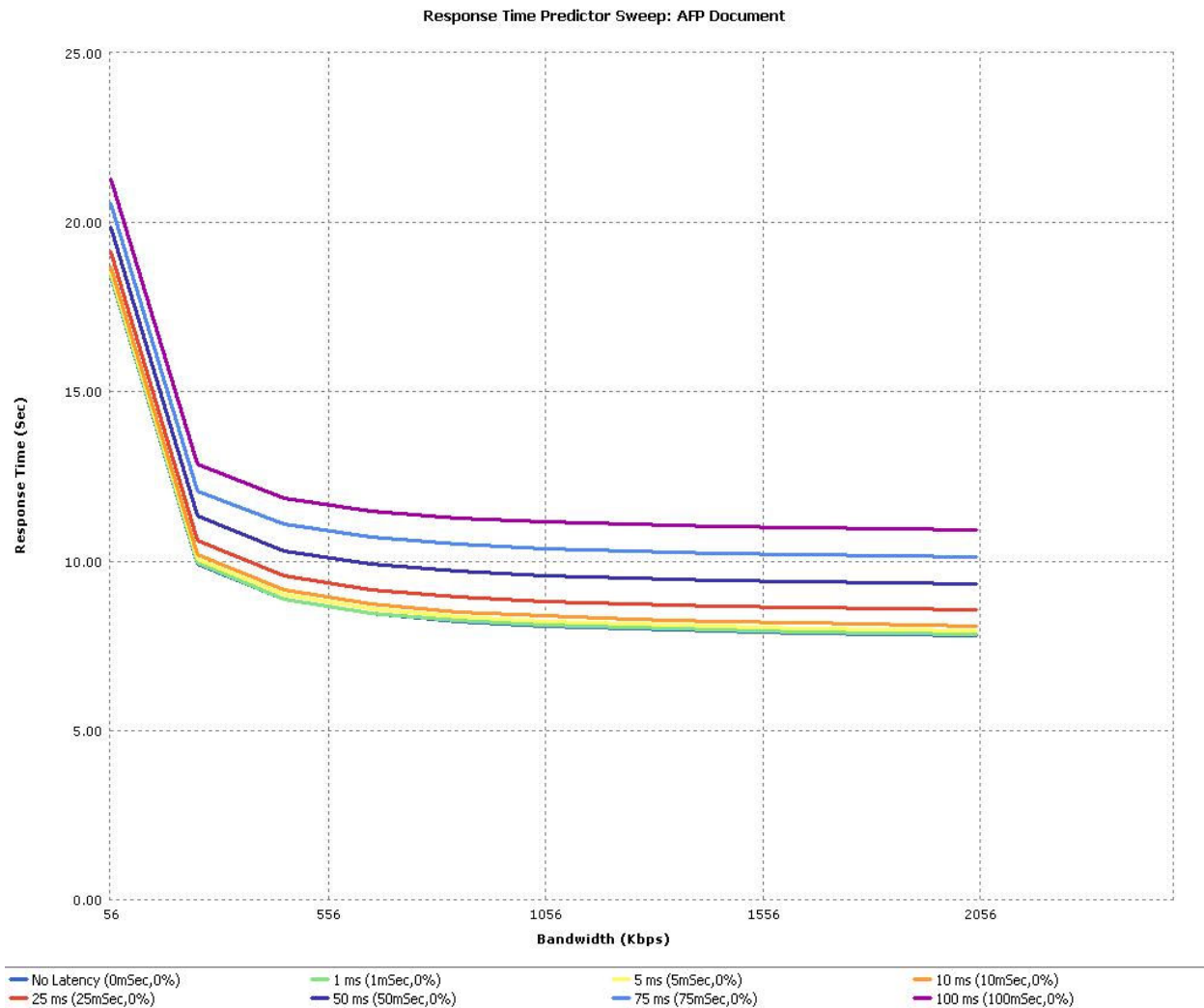


Figure 48 - Response Time Sweep of AFP Documents

Report Appendix A: Additional Resources

If additional assistance is required, Hyland's Performance Team can help. We performance test OnBase, assist with support, appear at performance testing workshops, and teach performance testing at TechQuest events. If you have a specific question, please ask your representative to forward it to the Performance Team for help.

We are also available for engagements to assist your testing, help you choose tools, or conduct performance studies onsite. Please contact your representative for more information.

Search engines, Performance Monitor, Task Manager, w3 and Event logs, the OnBase Diagnostics Console, and various other tools that can help you are generally already installed or available. You can learn much from these.

Windows Sysinternals is a collection of utilities to examine and troubleshoot low-level Windows activities. RegMon, Filemon, and Process Explorer are all utilities in this bundle:

<http://www.microsoft.com/technet/sysinternals/default.mspix>

Carefully observing the http conversations between the client and server is essential to constructing good load scripts. Consider using an http proxy such as Fiddler <http://www.fiddlertool.com/> to learn what these conversations look like.

A short overview of Performance Testing:

http://perftestplus.com/resources/requirements_with_compuware.pdf. Scott Barber has written a lot more about performance testing: <http://www.perftestplus.com/pubs.htm>.

Microsoft's Patterns and Practices Site has information on performance testing:

<http://www.codeplex.com/PerfTesting>. The performance testing guide is still in early Beta as of June 2007, but is as good a summary as available.

There is design guidance on the main Patterns and Practices site: <http://msdn2.microsoft.com/en-us/practices/default.aspx>

Rico Mariani blogs on performance from the architecture/coding side:

<http://blogs.msdn.com/ricom/default.aspx>

[Lessons Learned in Software Testing \(Amazon\)](#) is an invaluable resource for any tester. The application of context-driven testing principles can improve the effectiveness of any testing effort.

[Center for Software Testing Education & Research](#). An online course in software testing is now available from two testing thinkers we greatly respect, Cem Kaner and James Bach.

Report Appendix B: References

Microsoft. (2004, June 8). *Improving .NET Application Performance and Scalability*. Retrieved August 6, 2007, from Microsoft: <http://www.microsoft.com/downloads/details.aspx?FamilyId=8A2E454D-F30E-4E72-B531-75384A0F1C47&displaylang=en>

Miklos, T. (2006, November 20). *AIDA 32*. Retrieved August 6, 2007, from Sofotex: http://www.sofotex.com/AIDA32-download_L9326.html