

[Presentation](#)
[Paper](#)
[Bio](#)
[Return to Main Menu](#)

P R E S E N T A T I O N

T8

Thursday, October 26, 2000
11:30AM

STRESS TESTING LOAD ON A SERVER

Elisabeth Hendrickson
Aveo Inc.

International Conference On
Software Testing Analysis & Review
October 23-27, 2000
San Jose, CA, USA

Stress Testing Load on a Server

*(or, “Stuff I Wish Someone Had Told Me
about Load Testing Four Years Ago”)*

Elisabeth Hendrickson
Aveo Inc.

My Load Test History (abbr.)

- ◆ Early 1997: 5 concurrent users simulated using a GUI test automation tool and 5 client machines
- ◆ Early 1998: 50 concurrent users simulated using a proprietary command line interface
- ◆ Late 1998: Desire to simulate 100,000 users thwarted by reality



Overcoming Myths

- ◆ The tool does all the hard work
- ◆ One person can do all the work
- ◆ We don't need anything fancy
- ◆ It shouldn't take long to get the results
- ◆ We'll know everything we need to know after we run the load tests



The Server Environment

- ◆ IIS Web server handles connections with client software
- ◆ Proprietary server software handles business logic
- ◆ Database holds information sent to client's machine
- ◆ These can be on any number of machines for scalability

Current Load Test Challenges

- ◆ Server software includes proprietary and 3rd party commercial pieces
- ◆ Data goes up via HTTP, but in a proprietary format
- ◆ Data is encrypted
- ◆ No UI on the server; limited UI on the client

Establishing the Purpose of the Test

- ◆ Determine maximum capacity of the system
- ◆ Determine how two different server farm configurations help with scalability
- ◆ Identify the bottlenecks in the system
- ◆ Determine how well the system performs under load

Establishing Tools & Methodology

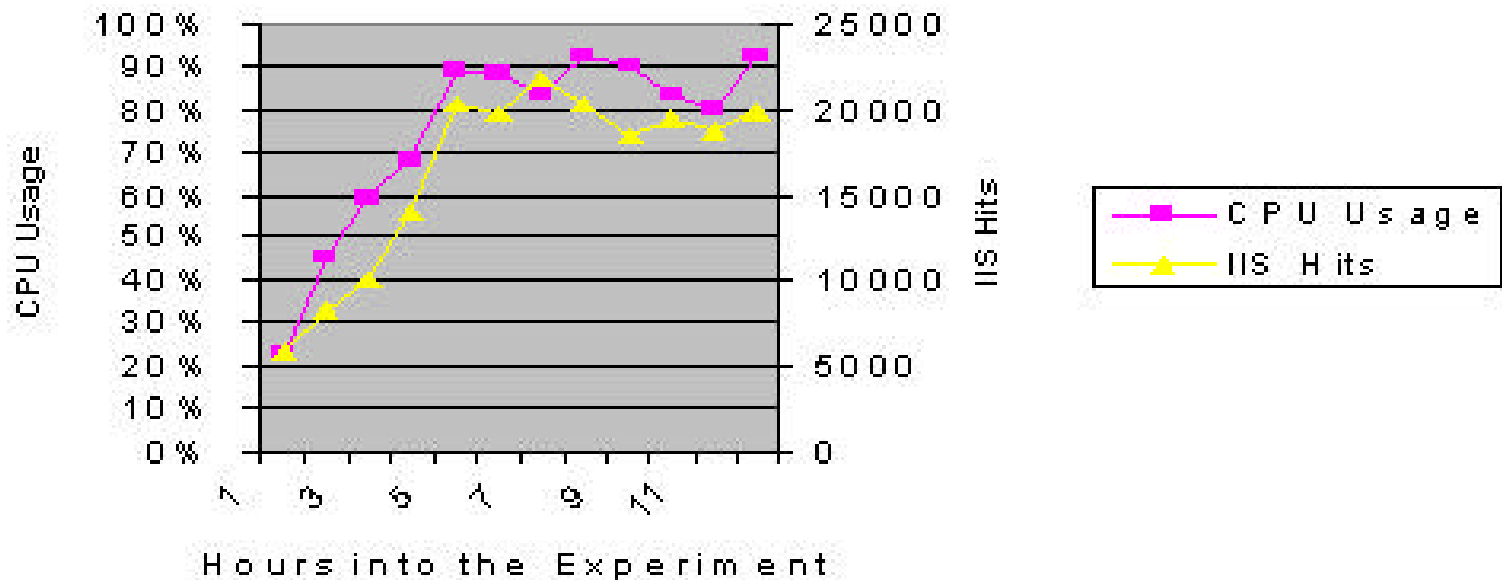
- ◆ Modeled the usage of the system in the real world to set targets for number of connections
- ◆ Debated whether to stress just our piece or the whole system—decided on whole system
- ◆ Chose test execution & monitoring tools: WCAT, enhanced client software, SiteScope
- ◆ Identified machine resources
- ◆ Chose test result analysis tools: Perl, Excel

Created Sample Reports

Avg. Connections/Minute

Avg Response Time (in seconds)

CPU Usage / IIS Hits



Executing the Tests

- ◆ Machines weren't always available
- ◆ Bugs not related to server performance hampered progress
- ◆ It wasn't as easy to get the scripts working as we expected
- ◆ It just took longer than we thought

Analyzing the Data

How do you summarize 1 Gigabyte of data? In stages, using tools.

Raw Data

```
2000-01-27T18:36:46-0800 1
2000-01-27T18:44:52-0800 1
2000-01-27T18:50:23-0800 1
2000-01-27T18:54:42-0800 1
2000-01-27T18:58:23-0800 1
2000-01-27T19:02:11-0800 1
2000-01-27T19:04:10-0800 1
2000-01-27T19:04:13-0800 1
2000-01-27T19:04:14-0800 1
2000-01-27T19:04:16-0800 1
```

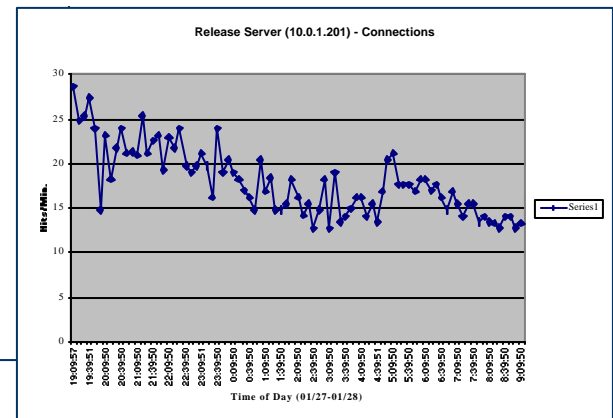
Perl

Tabular Data

```
2000-01-27 21 407
2000-01-27 22 374
2000-01-27 23 374
2000-01-28 00 323
2000-01-28 01 313
2000-01-28 02 289
2000-01-28 03 297
2000-01-28 04 299
2000-01-28 05 336
2000-01-28 06 372
```

Excel

Graphs and Charts



Results of Analysis

- ◆ Found and fixed bugs
- ◆ Simulated up to 25,000 connections/hour on our proprietary server
- ◆ Did not exceed the capacity of the server (CPU logs showed we weren't making the server work hard)
- ◆ Conclusion: Didn't meet all goals of the test, but still good to go
- ◆ Post Script: Server has held up well through dramatic growth in connections/hr

Key Success Criteria

- ◆ Cooperation with internal customers on setting goal of test and report format
- ◆ Support from development to create the tests and debug problems
- ◆ Sufficient resources (people, machines) allocated to the task
- ◆ Focus on continuous improvement, not all or nothing



Conclusions



- ◆ Design carefully
- ◆ Work with your stakeholders before you execute the tests
- ◆ Allow more time than you think you need
- ◆ View it as an “experiment” rather than a single “test”

Stress Testing Load on a Server

(or, “Stuff I Wish Someone Had Told Me about Load Testing Four Years Ago”)

Elisabeth Hendrickson

8/9/2000

In early 1997, at a loss for how to simulate multiple users through a Web-based client running on PCs, I reached for the only tool at my disposal: a GUI test tool with distributed testing capability. Although my experiments found multi-threading bugs and multi-user issues, my efforts seem rather paltry these days. I was able to simulate up to five (5!) users with coordinated automated scripts. In retrospect, it almost seems like it would have been easier to get five testers to play with the system at the same time.

On another server project about a year later, I helped create a suite of load tests using proprietary tools—and was able to simulate up to 50 users.

In late 1998, I expressed a need to a major tool vendor to simulate 100,000 users for another server project. I was surprised when a vendor representative told me that a 100,000 user load test would be one of the largest attempted in her experience. Ultimately we used a combination of proprietary and commercially available tools to simulate about a fifth that number—and it was a difficult and time-consuming project.

Even now, with more tools available than ever before, it’s difficult to simulate a large number of users. If it is so difficult to do large-scale load tests, how can anyone be sure that their server software won’t crash and burn under real world use conditions? This paper examines the difficulties of really stress testing the load on a server as well as tools and techniques to help solve the most common problems.

Myths and Realities

During the projects mentioned above, I encountered myths that interfere with designing good load test experiments. Before describing techniques for designing and executing successful load tests, it’s worth discussing common misconceptions about load testing.

Myth: *The tool does all the hard work. If we just buy the right tool we’ll get all the data we need.*

Reality: Maybe, but probably not. Define what you want to measure first, then see if the tool can provide that data. You will probably need additional tools to measure different aspects of the system and to analyze the data. Even if you do find a tool that provides all the information you need, you will need to interpret the significance of the results in your context. Don’t underestimate the difficulty of understanding the information you’ve gathered.

Myth: *One person should be able to handle all the load testing.*

Reality: Load testing requires a diverse set of skills: requirements gathering, test design, technical, and data analysis. You are unlikely to find all those skills in one person. Even

if you do find someone with all those abilities, it would take a long time for one person to get all the necessary data.

Myth: *We don't need anything fancy. Just throw a bunch of users at the system and see what happens.*

Reality: Whether or not you need anything fancy depends entirely on what you hope to learn from the experiment. You could choose to do a "live" load test with a lot of real users by launching a pilot or beta version of your server—and in some cases this is the right thing to do. However, load tests with simulated users have a number of advantages: you can control what the simulated users do to force overloads in different parts of the system; you can shut down the servers, reconfigure, and re-run the test to see how the configuration changes the results; and you can more accurately control the number of simultaneous connections.

Myth: *How hard can it be to simulate a whole bunch of users? Just write a program to do it.*

Reality: The short answer is that it can be very hard. In my experience, there are physical constraints on how many users you can simulate on each test machine. With the tools available these days, you should be able to do much better than my pathetic 5 users, or even 50 users. But simulating 100,000 concurrent users with non-trivial transactions is still daunting.

Myth: *It shouldn't take long to get the results—it's just an overnight test after all.*

Reality: It will take longer than you think (see the case study at the end of the paper for an example of how slips occur). It takes a lot of time to design the test, set up the machines, execute the test successfully, and interpret the data.

Myth: *We'll know everything we need to know the day after we run the load tests.*

Reality: You'll know everything your experiment was designed to tell you after you analyze the data. It can take a long time to analyze the data—longer than it took to plan and run the experiment. Even once the data analysis is done and the reports have been distributed, the experiment may or may not tell you what you actually hoped to learn; you will probably need to run subsequent experiments to answer lingering questions.

Designing the Tests

Management suddenly realizes, "We'd better find out what's going to happen if we're wildly successful!" And so they call you in to run some load tests. "Just load up the server with a bunch of users—say 100,000 or so—and tell us what happens," they say. "Can you get us some numbers by next week?"

You grumble under your breath on your way back to the test lab, "Oh sure. Next week. Just load up the server. No problem. I'll just whip these here 100,000 users out of my pocket."

Before you spend too much time trying to figure out how to simulate the requested 100,000 simultaneous users, make sure you understand your test requirements. Your management is probably looking for different information from what they initially requested.

Model the System

The first step in understanding your test requirements is to make sure you understand the parts of the system being tested, how it will be deployed in the real world, and how users will interact with it.

- What does the server-side architecture look like? Is there a database at the back end? Does logging go to a file? Are there proprietary elements, or all 3rd party software? Are you expected to provide information about how all the parts react under load, or just the parts created by your company?
- Will the application rely on only one server on the back end? Most large server-based applications run on "server farms" in which multiple servers share the load. Do you need to recreate the production environment? Or can you scale down the experiment by scaling down the hardware on the back end?
- Will users interact with the system as with an e-commerce Web site? If so, do you know how long a typical session is and what users most often do? Does it vary by type of user? Or will client software pull down information at specific intervals as with an email server? If so, do you know how much data users will pull down? Make sure you define different types of users and activities—you'll want your scripts to be truly representative of typical user behavior.
- How many users are truly likely to do the same thing at the same time? Does it make more sense to talk about simultaneous users, users-per-second, or users-per-minute? Are there peak usage times where you have to account for a larger percentage of the users accessing the system simultaneously?
- How much data will be uploaded or downloaded in real world situations? Load tests performed with a toy data set can result in misleading information about the capacity of the system if it turns out that the size of the data set affects the performance of the system. Use real data from the system currently in production if at all possible.

In one case, by using real production data to model the behavior of users on our system I was able to learn that during the peak times, roughly 1% of our user base was connected during any given minute. Using this information along with sales projections, we were able to choose an appropriate target number of connections per minute for our load tests—and it was significantly less than 100,000.

Understand the Purpose of the Tests

One common testing approach is the load up the server until it stops responding. These "fall over dead" tests are dramatic, but of marginal use. Knowing the breakpoint of the server is just one piece of information you could gather. There are a number of people interested in the results of load tests. Understanding what these stakeholders need is the key to understanding the purpose of the tests you're running.

Management needs to know how the system scales—when to buy more hardware and how to predict hardware needs based on projected number of users. They may also want to know the breakpoint: when performance problems will become performance crises.

However, their real interest is usually in getting information to help them plan and budget.

System Administrators need to know the most likely signs of stress and strain on the system so they can add capacity just before it's needed, not after. They also need to understand how the system behaves under load—does the system still work correctly when the CPU is pegged at 100% for extended periods of time, or do they need to make sure that never happens?

Developers need to know about bugs that interfere with the normal functioning of the server under load. At worst, the server should slow a little under load, not crash, corrupt data, lock out users unexpectedly, or produce incorrect results. The system may exhibit bad behavior under load—and the developers are counting on you to characterize the bad behavior well enough for them to fix the problem. Developers may also be looking for information about bottlenecks—are there specific areas of the system they can optimize to increase capacity?

As you are designing your experiment(s), check with each person interested in the results of the test to make sure your experiments will provide them with the information they most need.

Choose Your Measurements

If the goal of the load test is to characterize the behavior of the system under load, you need to take enough measurements to truly understand how the system behaves. One measurement isn't enough. At the very least, you should consider taking the following measures at short intervals throughout the experiment.

- A count of the number of concurrent users or connections on each relevant part of the system
- Response time for a pre-defined action such as a query
- A count of the number of errors (usually available through log files)
- CPU and memory usage

Your system may have additional meaningful measurements. The more information you can gather during the experiment, the better. Load tests typically take a long time to run. You don't want to have to re-run the experiment simply because you forgot to monitor a key aspect of the system.

Think of the system monitors during the load test like monitors on a patient at a hospital. Taking a pulse can tell you whether or not the patient is alive. Measuring heart rate, blood pressure, cholesterol, weight, etc. will tell you a lot more about the patient's current state of health.

Note that it's easier to say you want to measure something than to actually measure it. As you're determining what you need to measure, think about how you will gather that data. (More information about tools to help is available in the Tools section farther on.)

Design Your Reports

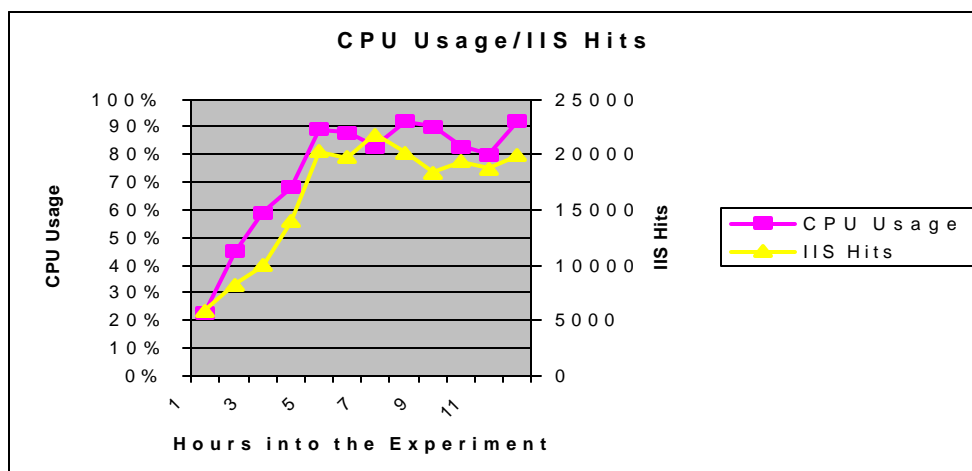
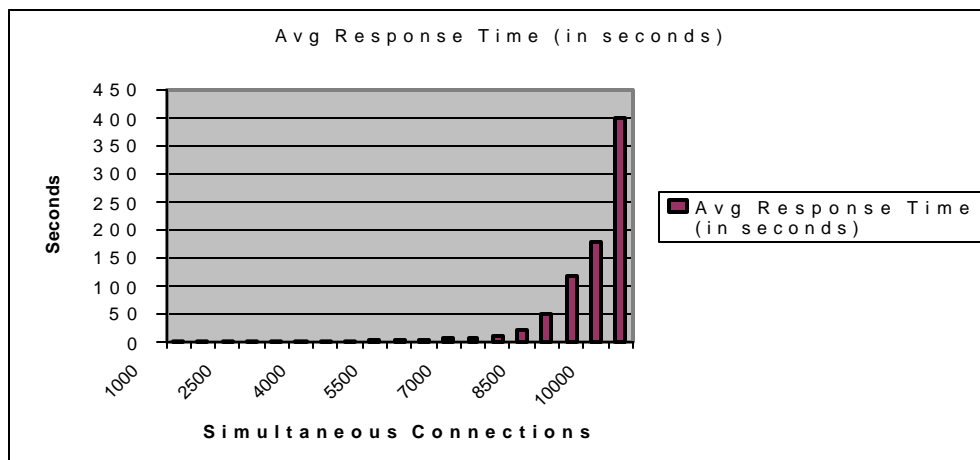
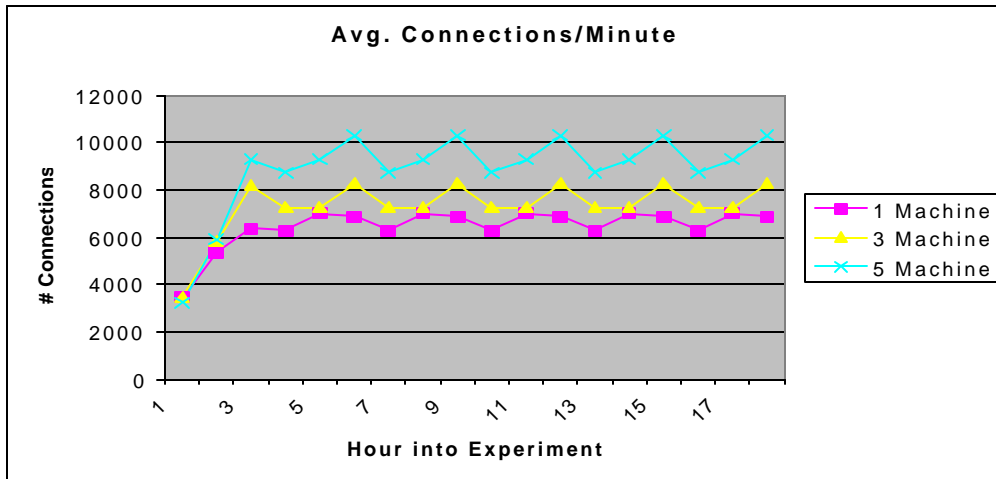
It's important to design your reports before you start running tests to make sure you're planning to gather the right information and present it in a way that your stakeholders can understand. It's frustrating to spend a week running tests and another week analyzing data only to discover that the reports were immediately relegated to the shredder because the stakeholders didn't care about or understand the report.

To prevent miscommunication about what information you're gathering and what it means, generate a fictitious report first that shows the charts and graphs you expect to produce at the end of your experiment. Circulate your fictitious report (clearly labeled as Sample Data—NOT REAL!) to all your stakeholders and ask them, "If I can provide you with the information shown on this report, will that meet your needs?"

For example, consider the following report:

Server Load Test Results Report SAMPLE

Not Real Data—Fictional Data Used to Demonstrate Report Format Only!



In order to produce reports like this, at least one member of the load test team will need be skilled at data analysis. If you or your teammates don't have much experience parsing through logs or producing charts and graphs, either find someone who does have this experience who can augment your team or get training in data analysis tools. The best load tests in the world will yield little or no results if you can't communicate the important data in a way your stakeholders can understand.

Preparing to Run the Experiment

Stock Up Your Toolbox

You may be thinking that you just need a tool to load up the system. Once you know how to put a load on the system, the rest is easy, right? You actually need several kinds of tools. At a minimum, your virtual toolbox needs to include:

- A tool to simulate a large number of connections or users. Examples: proprietary tools using a command line or programming interface, WCAT (distributed as part of the Microsoft Developers Network), and commercial load test tools from Mercury, Segue, Compuware, etc.
- A tool to monitor the RAM and CPU usage on the server(s) every few seconds over the course of the experiment. Example: SiteScope
- A tool to monitor the health of the system under test. Example: real clients scripted to perform actions throughout the course of the test and measure response time
- One or more tools to parse verbose logs into raw numbers for analysis. Example: Perl scripts
- A tool to analyze the data for patterns and produce charts and graphs. Example: Excel

Identify Machine Resources

Unless you have the luxury of a dedicated set of machines for load testing, you will probably need to arrange to use machines well in advance of the actual experiment.

Start by identifying your needs:

- **How many servers do you need for the software under test?** Will you use a single server or recreate the production environment server farm? Plan to run the server software on dedicated machines—do not plan to run the load test tool on the same machine as the server software since the additional load on the machine could skew the results.
- **How many other machines (clients and servers) do you need and in what configurations?** Be sure to check machine requirements for your load test tools: most tools can simulate only so many users per machine, so you'll probably need multiple machines running the load test tool to achieve the number of simulated users you intend. Note that most load test tools also require specific operating

systems (NT Server, for example). Will you use machines just to run the load test tool or do you also need a pool of machines available to run the real client?

At a bare minimum, you will need two machines: one to run the server software under test and the other to run the load test tool. To do a full-blown load test experiment, you're likely to need 5 – 20 machines, or more if you need to do high volume load testing.

The best way to communicate about your machine needs is to map out the lab setup you need. Draw a picture that includes each machine you plan to use, the purpose of the machine, the system requirements for each machine (OS, RAM, etc.).

Line up your machines early—get permission from the people who control the machines you hope to use well in advance of your actual experiment. Set their expectations about when you will use their machines and for how long.

Run Your Experiment

There's a great feeling of anticipation in the air at the beginning of a well-designed load test experiment. Will the experiment yield useful results? Will it run to completion? Will the server be able to handle the load?

It's a good idea to ramp up the experiment slowly, bringing real and simulated clients online a handful at a time rather than all at once. This serves two purposes:

- In the real world, you usually don't see an all-or-nothing traffic pattern. The results from an experiment where you ramped up the number of connections over time is likely to be more realistic than if you bombard the server at the beginning of the experiment.
- If the server can handle a lot less than you thought, you'll see the signs of stress and strain in the results before the breakpoint. This will give you a better feel for where the breakpoint is than if you throw everything you have at the server at once and it falls over.

As you bring your clients online, remember that almost nothing goes 100% according to plan. The goal is to get the data, not run a perfect experiment. It's OK if one of the machines dies and has to be replaced. It's OK if you realize mid-way through that your test server is also hosting a coworker's clandestine e-commerce web site and therefore is getting extra, unanticipated traffic. Just keep going and take careful notes—you never know what might be important later.

Since your goal is to gather information about how the server performs under load, it can be annoying when bugs in other parts of the system interfere with your experiment. You may find bugs you never anticipated in the client software or in parts of the system that you didn't think you were testing. As annoying as it can be, these bugs are just as valid as the bugs you expected to find. Try not to dismiss them as irrelevant or mere impediments to your success.

At the end of the experiment, gather all the data you can get your hands on from the server and the client machines: all the log files and output files from all the tools you used and from the software under test.

Analyze Your Results

After one set of experiments, I had nearly 1G (yes, one gigabyte) worth of data to analyze. When you're dealing with thousands of pages of logs, there's no point in even trying to identify patterns or glean knowledge simply by reading the output. The nuggets of information will only be apparent after doing some analysis.

So how do you analyze results? Use a tool to parse through the logs, pulling out information like number of connections over time, errors over time, CPU and memory usage over time. Then use a tool to graph that information—people understand pictures of data much better than raw numbers.

The important thing about analyzing your results is to explain the significance of your findings. Don't just dump a whole lot of data into your stakeholders' laps and expect them to be able to figure out what it all means. By this point, you understand the peculiarities of the system under load better than your stakeholders. Analyzing the results is the key to sharing that understanding.

Case Study: One Load Test Experiment

One experiment provides a particularly good case study of some of the issues discussed in this paper so far.

Day 1: off to a good start—we met with the people most interested in the test results to make sure we all had the same goals. We determined that the goals are:

1. Determine maximum capacity of the system.
2. Determine how two different server farm configurations help with scalability.
3. Identify the bottlenecks in the system.
4. Determine how well the system performs under load.

We determined that we would use a combination of WCAT scripts and real clients to put the load on the server. The real clients are enhanced so they will automatically connect more often than usual, adding even more load to the system.

Day 2: Houston, we have a problem. The scripts aren't working.

Day 5: Finally got the scripts to work. Doing a trial run overnight to make sure everything will work unattended.

Day 6: The scripts are working, but the enhanced clients stop responding at midnight. Looks like a bug in the client. We need to get that fixed before we can really do the experiment.

Day 9: Having trouble reserving the machines in the lab—the other testers are resisting having the machines tied up from 5pm to 10am to run the tests.

Day 10: Finally able to execute a test through to completion.

Days 11 – 14: Executing tests for first configuration.

Days 15 – 18: Executing tests for second configuration.

Day 19: The series of tests produced a LOT of data to sort through. Analyzing...

Day 31: Results report submitted.

Results of the experiment: we simulated up to 25,000 connections per hour, less than we'd originally hoped. However, the server easily withstood the load (as shown by CPU and memory usage), so we decided to release. Although the experiment did not achieve all 4 goals, it achieved enough of the goals to make a decision. And the goals are not lost: the design work we did on this experiment will be reused on the next experiment.

Conclusion

If you do a good job of designing and executing your load tests, you're likely to find interesting bugs. In one experiment (not the one described above), we produced a graph that visually demonstrated how the server handled steadily fewer connections over the course of 6 hours until it completely stopped responding. The problem? Threads were being spawned but never freed. Eventually the server software was unable to spawn more threads and it hung.

We never could have found the bug without doing a load test—and it would have been a disastrous bug to ship with. Showing the trend in number of connections handled per minute made the behavior clear to even the least technical member of the project team. Clearly this was a bug we needed to fix.

Most people seem to put the vast majority of their effort into finding, learning, and using tools. For any given load test there are probably at least three or four ways to get the test done. The hard part is figuring out how to design the test well so that it gives you the information you really need. I hope this paper has helped you with that hard part, at least a little.

Tools & Bibliography

SiteScope is a tool for monitoring the health of servers: <http://www.freshwater.com>

WCAT is a tool from Microsoft for load testing web applications:
<http://msdn.microsoft.com/workshop/server/toolbox/wcat.asp>

Perl for Windows is available for free download from <http://www.activestate.com>

Anderson, Mark D. "The Top 13 Mistakes in Load Testing Applications," *STQE Magazine*, October/November 1999.

Elisabeth Hendrickson

Elisabeth Hendrickson (ehendrickson@aveo.com) is the Director of Quality Engineering at Aveo Inc., and she has bought—and built—numerous tools. More of her articles are available at www.qualitytree.com.