Performance Testing E-Commerce Web Systems
Presentation Paper

Mike Hagen
The Vanguard Group
5/3/2000

Slide1 – Presentation Title
Slide2 - Opening

Yesterday, your marketing department started an ad campaign that told everyone to come to your Web site. Today, your Web server is down due to everyone doing what they were told. Tomorrow, the Wall Street Journal has an article on the front page saying how your site was down all day. Do not let this happen to your Web site.

By performance testing your Web systems before they are elevated to production, you will know in advance what needs to be beefed up for the "big day". At Vanguard, our biggest day happened over the Y2K weekend when many of our clients logged in to check on their life savings. This presentation will go through how we prepared for this event and how you can do the same.

There are two ways to predict how your systems will hold up under a lot of load, Performance Testing and Modeling. Modeling uses software to simulate entire systems and allows you to do 'what if' scenarios without having the actual resources. Modeling is an extremely valuable tool that can, and should, be used in conjunction with performance testing. Performance testing runs actual applications on physical systems with monitoring devices in place to report on how the system is performing. The tests should be executed on 'production like' systems that have controlled software. Setting up this testing environment will be the first thing you have to do.

Slide3 – Setting Up a Performance Test Environment

Setting Up A Performance Test Environment

You cannot get accurate results without an accurate testing environment. It has to match your production environment as closely as possible. This is a difficult and expensive task. But if you have a 20 CPU E6500 web server running Apache in production and you are testing with a SPARCserver5 running NAS, you are not going to get the same results. Making sure the software matches up should be easier than obtaining the hardware. So I will go into the hardware first.

Hardware

How do you justify spending a million dollars on a testing environment? You gather up all of the articles from the last holiday season, when most of the E-commerce sites failed to handle the onslaught of shoppers, and plop them on the procurement manager's desk. If you work for a company that thinks like most other companies, you have to use dollar signs. If you just try to use common engineering sense, you will fail.

The ideal test environment would be an exact replica of the production environment. Undoubtedly you will not be able to spend all of the money that you want and you will have to find ways of stretching your budget. One way to save some cash is to build a scaled down version of production. Meaning if you have 20 web servers in production, buy 5 for testing purposes and extrapolate the results. The hardware has to be the same, but you do not need to have as many. The best way to go about determining how to scale your system is to look at how your load is balanced in production. For

example, if traffic is round robined to different threads of hardware, then you could buy one thread and scale up the results by the total number of threads.

Once you have determined the scalable layout of your testing environment you will need to make sure every component (networks, firewalls, servers, mainframes…) matches the production equipment. The weakest link theory applies here. You could have all of the server power in the world but your end to end response times could be long due to an undersized firewall bottleneck. You will also have to make sure the hardware is configured the same way. For instance, if your firewall in production has 40 rules, the test firewall has to have 40 rules. All of this little stuff really adds up when you are talking about the extremely high level of traffic on today's Web systems.

Next you will need to put in the request for more hardware to be used as your physical load generators. The physical generators are the boxes where the simulated Web user traffic will be spawned. I will go into these simulations later. You will need enough processing power and network bandwidth to support the amount of traffic that would normally be coming from you ISPs. A regular PC probably cannot support this kind of traffic. You will need a bank of PCs or maybe one good sized Unix box. Plus the network connection from these load generators to the Web systems needs to be sized properly. A 10 Mbit Ethernet connection is probably going to get maxed out. If you have a T3 connection to your ISP in production you could get ~45 Mbits of traffic. Therefore, your test network will have to support at least that and hopefully more in case you want to do capacity planning tests for more ISP connections.

Software

Configuration management is so important in testing efforts. Hopefully you will have a team dedicated to this. But if you do not, you will have to ensure that the versions of software that you are testing are what will be going into production or are already in production. Take the time to make a list of every piece of software in your system (operating systems, communication software, web server versions, etc.). Make the same list of the production systems and make sure everything that differs is either changed or slated to change once you go to production.

This also holds true for system configuration files. Settings like logging levels and server timeouts can really invalidate your test results if they are not configured like they will be in production. Ideally, the config files used during test should be directly copied into production with only a few changes to some global variables. If you can do this, you are probably in good shape.

Slide4 - Determining Web User Traffic

Generating Web User Traffic

Once your testing environment is all set, you will need to come up with a way to generate the massive amount of user traffic that web systems realize. The old method of mustering up everyone in the office for a big performance test will not cut it anymore. You will need to use software to simulate the load. There are a number of tools on the market that do this. Examples would be Mercury's Load Runner and Radview's WebLoad. My experience is with WebLoad.

WebLoad uses JavaScript agendas to simulate users walking through your Web site. WebLoad has an authoring tool that records the server requests that a browser makes. Using this tool you can create many agendas that hit pages all over your site, including secure pages. You do not necessarily need to create scripts for every page on your site but you should cover the most frequently hit pages as well as pages that have known performance problems. The scripts should be instrumented with page timers and error checking. For instance, we put a timer around every Get or Post. You can display these timers on the Webload Central Console as the test is running and export them to Excel for more detailed reporting. Since the pages will not actually get rendered in a browser for you to see, you will have to come up with a form of error checking that ensures that the actual page content that you expected to be served is what was actually served. The way we handle this is to parse through the HTML of every page that is returned, looking for key HTML text. An example of this is checking for the <TITLE> tag in the html. If the TITLE of the Account Balance page comes back with anything other than Account Balance, an error is generated. This increases the amount of work that the generators have to do, but it is necessary. Otherwise you might be getting blank pages with no indication that anything is going wrong.

Slide5 – Java Script Example

After the scripts are created, the next step is to figure out how many users should be simulated running through the various scripts.

Slide6 – Determining Web User Traffic (cont.)

Determining the projected peak amount of users that will be hitting your homepage during the busiest time of day is not simple. Hopefully you will have a marketing team that has these projections. If not, you will need to get peak usage information from your production system and project to some logical increase or decrease. You use these projections when it comes time to generate the load on the system. You will also want to choose a representative distribution algorithm (exponential, constant, Poisson) to simulate how the load enters you system. It is probably not accurate to have all of your users arrive at a constant rate every minute or so. What you will see is a big spike across the board every 60 seconds.

With Webload, you create load templates that define which scripts are to be run on which logical generators and at what load. A load template will be comprised of multiple logical generators. One logical generator may run your “Homepage” at a load of 250 users. Another logical generator may have your “Registered User Login” script running at a 400-user load. Depending on the scope of the test, you may have 50 scripts rolled into 1 template. Then, when it comes time to run the test, all you will have to do is start the template, not each individual script.

When you have an accurate representation of a production system, the hardware to generate the load, and the Web user activity scripted, you are ready to run the tests.

Slide7 – Running and Monitoring Tests

Running and Monitoring The Tests

At this point you will have spent a great deal of time and money for this wonderful thing called Performance Testing. Management will be looking for some justification. So you better show some real results. Just kicking off the load generating tool and watching the servers die is not going to make upper management buy more hardware. They are going to want to know what piece of code is making what piece of hardware choke and why. Why is the hard part. You will hear statements like, "You mean to tell me that I have 12 engineers in this lab and no one can tell me WHY the response times just jumped!" To get this information you need to have as much logging and monitoring as possible, without causing your own performance problem. If 20 people are 'grepping' through access logs during a test, you are going to see the CPU usage skyrocket. Likewise, if you set the level of logging too high, your servers will be spending more resources reporting what they are doing versus just doing.

Let us look at the load generators first. Depending on the type of testing (stress, load, endurance), you will have to choose a load template that represents the functionality and level of use to be executed on your Web system. A load test of the entire baseline of software is one of the more involved tests and will require a lot of different scripts to be run at the same time. The load generators will be sending and receiving requests at very high rates. They will also be reporting on the most important performance measurement in Web systems, end user response times. WebLoad has a Central Console that controls the running of the test.

Slide 8 - Webload screen shot

The console is used to start and stop the test, to monitor the test from a browser's perspective, and to collect all of the end user response time data. The load template shows up in a separate window and displays the physical and logical load generators, and all of the scripts that are running on the generators. The other windows display graphical and spreadsheet representations of the test data and any error or informational messages. This screen shot shows what happens when good code goes bad. Under load, this application could not take the heat and it took another application running on the same server out with it. Notice that other functionality is going along smoothly. These pages happen to be served from a secure web server, where the other troubled pages are coming from a public server. This demonstrates a perfect baseline load test scenario. Alone and/or at low load, all of these pages may have worked fine, but together they broke each other and brought down a Web server. It also demonstrates the need to test a variety of your Web pages. If we had not scripted this one page, we would not have caught this glaring problem and we would not have seen it until it hit production and we lost our public Web server.

Slide9 – Running and Monitoring Tests (cont.)

During testing it is extremely crucial to have some type of monitoring on all of the servers responsible for creating the web pages. If there are UNIX servers you can run

command line performance monitors like vmstat or /usr/ucb/ps. Another option that we use at Vanguard with our UNIX servers is a monitoring tool called TeamQuest. TeamQuest provides invaluable insight into server CPU, memory, and I/O. It uses fairly passive monitoring engines on your UNIX boxes and reports back to a central workstation that can graphically display resource usage at the process level. If you are in the middle of a test and all of a sudden you get a big spike in CPU on your application server, you can drill into the spike and get a listing of every process running at that point. It will also give you who are running the processes. For instance, with a big spike in CPU you may drill down to find that user id mhagen issued a 'grep' on a 20Gbyte logfile. After revoking his access to the box, you can then continue with your testing. If you are monitoring free memory and you see it declining over time, you probably have a memory leak in your code and need to drill down into the individual processes to see what is the culprit.

If you have a Mainframe in the loop, there are a number of ways to get transaction information. Omegamon seems to be a fairly universal tool. It allows you to drill down to individual transactions and find CPU usage, amount of I/O, and response times. Mainframes store lots of historical data. You just need to figure out how to get to it.

Slide10 – TeamQuest snapshot

Slide11 – Running and Monitoring Tests (cont.)

If you are an NT shop, the only tool that I have used is the generic Performance Monitor. It gives lots of overall resource information and allows you to log the data. But, if you need stats on individual processes, you will have to use the NT Task Manager. However, I am not aware of a way to log what Task Manager sees. I am sure there are many NT performance tools out there; I just do not have experience with them.

Network monitoring will let you know if your routers, firewalls or switches are causing bottlenecks. Many network sniffers are available and can be turned on and off quite easily. Do not let your telecom group tell you that there is plenty of bandwidth. Web sites generate an enormous amount of data. During peak moments, the network cannot be re-transmitting or dropping packets. Things will snowball quickly.

The last form of monitoring that I will mention is application log monitoring. If your code is not already instrumented with calls to an application log, put in the request now to have it done. If a CGI script has to make a database request, it is extremely helpful if you have a log of the round time for the request. If this kind of info is there, you can just tail the logs during the test to see how things are going. There is a programming standard called ARM (Application Response Measurement) that details the APIs for applications to log what they are doing or waiting for. This is wonderful. Tools like Tivoli's TAPM, use the ARM API and gives incredible insight into what your applications are doing. The instrumentation is usually designed with the ability to turn it off with a global application variable. That way, all of the excess logging does not occur when you do not need it and additional resources are then made available. Weather you

use homegrown application logging or a standard method, these logs will give you the data needed to answer the "Why is this not working?" questions.

Slide12 – What To Look For

What To Look For

Your first indication that something is wrong is the end user response times start to climb. Knowing which pages are failing will help you narrow down where the problem is. If it is your homepage, your public Web server is probably the place to look. If it is your product description page, go to the servers responsible for requesting and serving that kind of information. This is when all of the real time monitoring comes in handy. Check the CPU usage on the boxes. If something goes wrong somewhere, it usually takes the CPUs with it. Then determine if all of the hardware is functioning. Various utilities allow you to quickly run device inspections. If the hardware checks out, go to the software.

Software core files or abends are the most pronounced way of letting you know that something went wrong. Core files usually get written to known directories and mainframe abends are usually pretty pronounced and easy to track down. If nothing is flagrantly showing a problem, you may need to turn up the logging to capture more diagnostics or perhaps run debugging versions of the code to capture memory leaks and other not so apparent problems. I would love to be able to tell you how to troubleshoot every problem, but it is not that easy. However, the more you test and the more you monitor, the more you will learn about your system and why it might be acting funny.

Typical Trouble Areas

Slide13 – Testing Pitfalls

Testing Pitfalls

Load Generators

Every performance engineering group that I have dealt with underestimated either the processing power or network connectivity of their load generators. You need to look at the performance of these machines the same way you look at the Web systems they are loading. Nothing is worse than telling a developer that their code is bad and then finding out that it was your load generators causing the pages to come back slowly. That developer will question your findings from that point on.

Network Firewalls

It is the first thing that the user traffic hits. If it cannot handle the load, you are not going to be able to find any other problems down the line. Internal company firewalls usually have a lot more rules and a lot less CPU power than a production Web system firewall. The more rules that each connection has to pass the more CPU it will require. Use the same rules and the same hardware as production.

Slide14 – Testing Pitfalls (cont.)

Software Control

Know everything that is in your environment. If you just start linking in applications without knowing what has changed, you are setting yourself up for hours of unnecessary troubleshooting. Like politicians appending congressional pay raises to Farm Aid bills, developers will try to slip in all sorts of changes to the advertised upgrades. You will here statements like, “We just added a couple of images, they shouldn’t have effected your test.”

Test Data

Test data includes information like user account preferences, product information, and passwords. If this data changes without you knowing, your test scripts could fail. If possible, limit the access to this data to a bare minimum. Sharing a username and password database with another group WILL result in them accessing and modifying user information to the extent that it messes up your tests. It is inevitable. So lock this info down.

Slide15 – Testing Pitfalls (cont.)

Missed Pages

You may think that you have an adequate sample of the pages on your Web site. You run performance tests, everything was fine and they elevate the changes to production. Three hours later your site crashes because 3 people hit an obscure ‘Help’ page that happens to use 100% of a CPU on a 3 CPU machine. There are 2 ways to prevent this; Script for every page on your site or have a way for folks in your company to report on when they happen to stumble on a page that seems to be taking a long time to come back. The latter is kind of crude, but effective.

Unmonitored Resources

Everything is suspect. If you do not monitor it you could spend months chasing a problem that could have been caught by simply logging into a box and tailing a log file. Monitor everything!

Slide16 – Design Issues

Design Issues

Systems Designed To Handle Today’s Traffic

I read a great article by John Gantz in the February, 2000 issue of ComputerWorld magazine on capacity planning. His best quote was, "Your capacity planning should be based on your capacity to order systems, storage or memory and to get them installed. You'll be ordering on demand. " He went on to say that you should, “Choose an architecture that can scale to at least 20 times what you really think you'll need in six months.” This issue is what is killing most Web sites out there. Who knew that 50 million people would be checking stock prices 20 times a day?

Database Requests

Request for data can severely slow down the building of a Web page. We have found that making 1 big request versus many little requests reduces the overall request

time. This is due to the overhead associated with making a connection to database server. Data caching and replication are methods to remedy this issue and should be used whenever possible.

Slide17 – Design Issues (cont.)

Page Size & Number Of Objects

Once a Web page leaves your web server and hits the Internet, there is little you can do to get it across the continent faster. A 175 Kbytes page is going to take more then 10 seconds to get across the country over a 28.8 connection no matter how fast you build it. The obvious solution is to reduce the size of the page. But another solution is to reduce the number of objects on the page. Objects include GIFs, frames, and base html code. Each object requires a separate TCP/IP connection from the browser. If there are 3 frames, 6 different images and the base page, 10 connections will have to be opened. Today's browsers only support up to 4 simultaneous connections. Therefore you will see the first 4 objects load, then the next 4, then the last 2. You can reduce the number of objects by combining the GIFs into image maps and getting rid of frames all together.

Lack of Performance Requirements

Designers and developers need to know what their limits are. Simply telling them that a page has to download in 10 seconds is not going to mean anything to them. You need to come up with specific time and resource budgets for building a page. There should be requirements for page content, database accesses, page assembly and resource usage. Resource usage is needed to control scalability. You cannot allow an application to use 1 Gig of memory to produce a single page.

Slide18 – Design Issues (cont.)

Get the Latest Technology

Web system software from 5 years ago probably won't hang in today's environment. The latest application servers allow you to keep persistent connections to databases and share resources across multiple servers. The newest version of the mainframe operating system OS/390, now has a new TCP/IP stack to support Web type traffic faster. Distributed content servers now let you put your Web pages out closer to the users, therefore reducing the page download times. With the Y2K freeze now lifted, new software is going to be coming out all over the place. Keep your eyes open.

Slide19 – Questions