

# Neustar® Webmetrics®



White Paper | Best Practices for Load Testing: Do Try This At Home

## Best Practices for Load Testing

By the Neustar® Webmetrics® Professional Services Team:  
Joel Weierman, Steve Thurner, Anu Sandhanam, Jason Paddock and Mike Edwards

## Contents

Best Practices for Load Testing .....	1
Summary .....	3
Planning Your Test .....	3
When and how long to test? .....	3
How many tests should you run? .....	3
Real or Virtual Browsers? .....	4
How much load should you test with? .....	4
Calculating Concurrent Users .....	4
Test Data and Server/Network Monitoring .....	4
Configuration .....	5
Where do you expect your traffic to come from? .....	5
How should you apply your user load during the test? .....	5
Where do users spend the most time on your site? .....	6
How long do users take when navigating your site? .....	6
Do you use third party vendors? .....	6
What Makes a Good Script? .....	6
Content Validation Combined with Proper Use of 'WaitForPageToLoad' .....	7
Setting Proper Timeouts .....	7
Optimization of Locators .....	7
Pause Times .....	7
Randomization .....	7
Test Execution .....	7
Execution checklist .....	7
Successful transactions .....	8
Page views .....	8
Page load times .....	8
Errors .....	8
Throughput .....	8
Analysis .....	9
High-level results .....	9
Digging deeper: load times .....	10
Digging deeper: errors .....	10
Tracking down errors .....	11
Examples .....	11
Other Considerations .....	12
In Closing .....	12
About Neustar® Webmetrics® Load Testing Services .....	12

## Summary

Why load test? Simple. To determine how many customers your website or application will support— before you find out the hard way, with degraded performance or an outage, usually when it's least convenient.

Quick tidbit: More often than not, people greatly over-estimate their capacity. Many are shocked when they get their first set of load test results. They realize there's a lot to do to prepare for a full production release. To make all that work less daunting, this white paper breaks load testing down into the following key milestones and activities:

- Planning
- Configuration
- Scripting
- Execution
- Analysis

By following the best practices we've developed over hundreds of tests, you can make your load tests smoother and more productive. Even better, you'll help protect your brand and bottom line. With so much at stake, proven methods can make a big difference.

## Planning Your Test

### When and how long to test?

As you launch a new application or prepare for peak Web traffic—say the winter holidays are coming and you're enhancing your shopping cart—load testing should be part of your overall project plan. The ideal time to run your first test is after User Acceptance Testing (UAT) is complete, but prior to moving the application into production. Based on our experience, you need to allow for an absolute minimum of 2-3 weeks for planning, scripting and execution. Many people try to compress this time, but believe us, there are no magic tricks (none that don't result in a lot of heartburn and sleep deprivation). It's an incremental process, requiring careful review and time for your development team to address any problems.

### How many tests should you run?

Based on our work with hundreds of customers over the years, we've found that three is the magic number.

- The **first test** provides you a benchmark and snapshot of your site's current performance. This will always yield recommendations for improvement.
- The **second test** validates all the changes made after the first test and identifies any remaining bottlenecks. You'll probably make additional tweaks.
- The **third test** is your application's final pre-launch validation. Ideally, you'll uncover its peak performance point.

Of course, your mileage may vary here. Some people can complete this process with just two tests, but we've found it's best to start with three and run additional ones as needed.

## Real or Virtual Browsers?

Thanks to cloud computing, it's now cost effective to load test a website using real browsers. Real browsers let you create a more life-like load profile, unlike virtual browsers that simply mimic a browser's http request/response sequence. This is especially true when testing AJAX and Flash®/Flex®. With real browsers, you'll also be able to script your tests faster. In most cases, we highly recommend going with the real thing.

## How much load should you test with?

The answer comes down to whether you need a stress test or a load test. The former will find your application infrastructure's breaking point, exposing traffic bottlenecks. The size of the test is based upon the expected total amount of infrastructure traffic. The rule of thumb: approximately 250 concurrent users for each CPU core that processes front-end Web server requests. (This can vary depending on many factors, but 250 is a good starting point.) For example, if you had a new application with 2 front-end Web servers installed on a quad-core machine, a reasonable load would be 2,000 concurrent users (250 users x 2 machines x 4 cores).

In contrast, a load test reveals the volume of page views or transactions your site can accommodate. You can obtain this type of information from Web analytics data such as Omniture or Google Analytics. Generally, the target volume is based on the application's peak hour of usage. For example, say 50,000 pages are loaded in your peak hour and you expect a 20% increase in traffic in the next few months. Think about targeting a total load of 75,000 pages views, so you can determine the app's ability to serve up to and above the target page view count.

## Calculating Concurrent Users

It's helpful to use a calculator to determine the number of concurrent users needed to reach your target load. First, though, let's explain the difference between a "concurrent user" and a "unique user." A concurrent user runs through a transaction from start to finish, then repeats until the test is over. A "unique user," on the other hand, is simply a single execution of a concurrent user or the completion of one transaction (execution of the test script from start to finish). Depending on a number of factors, concurrent users are able to generate the traffic equivalent of thousands of unique users over the course of a standard test.

Say you have a 10-step checkout process and you want to ensure the site can handle 1,000 checkouts in one hour. That's 10 steps x 1,000 transactions = 10,000 page views. Let's also assume that we expect each page load to take 5 seconds and the user to wait 5 seconds on each page. This would give us a total session time of 10 steps x 10 seconds = 100 seconds. Plugging in 10,000 pages per hour and 100 seconds for the session time plus the desired test time (we generally recommend 60 minutes), you get a concurrent user count of approximately 278. Now you know how many users are needed to complete your key transactions.

Visit [blog.webmetrics.com](http://blog.webmetrics.com) for a simple calculator that approximates number of concurrent users.

## Test Data and Server/Network Monitoring

To avoid errors during testing, be sure to create an accurate data set. This can be time-consuming, so start early. It often means creating individual user ID's for website login or a selection of products available on your site.

Also, make sure you have a plan to monitor everything supporting your application—network devices, Web servers and application and database servers. Load testing will give you all the key front-end metrics. To get the most out of your test, you'll want to cross-correlate this information closely with the performance counters from these various components.

## Configuration

With your planning firmly in place, you're ready to start load testing. So what's the best way to configure your test? Let's look at all the pieces.

### Where do you expect your traffic to come from?

Whether this is a new or existing website, you probably have a feel for where your visitors will be coming from. A site like Amazon will have visitors from around the world, whereas the government site for a province in Canada will see mostly local traffic. The point is, when running your load test you want to have users coming from various access points, attempting to emulate your normal traffic. For example, if you were looking to get more users from Asia, it would be worrisome if you saw the pages taking twice as long to load from that access point. All this data will help you forge a better user experience.

If you don't have the capacity to apply load from the most important locations, you can also test from other far-flung locations. For example, if you need to add Tokyo to get the load you want in your test, here are a few steps you can take. Find the page load time from Washington, D.C (let's say it's 2 seconds when there is no load on the site). Then test from Tokyo (let's say the page load time is 4 seconds with no load). This lets you know that Tokyo has a 2 second lag, which can help you better interpret results from your Tokyo access point.

### How should you apply your user load during the test?

With a number of users per core in mind (likely in the neighborhood of 250) you need to apply that number to your load test. Web traffic usually comes in waves, surging and receding over the course of a day. Even with a major event like a new Super Bowl commercial, the surge is not instant—1000 users may appear over time, not in the same second. The best practice here is to set up your load test with intervals of ramps and constants, giving you granular data. A ramp is a period when we increase / decrease users, while during a constant the user count remains unchanged, but the users are executing scenarios on the site. If we take a 1,000-user test, we could set up an hour test with intervals like this (see chart to right):

This basically shows 20 total intervals, 10 ramps and 10 constants. During each ramp, users are added until the Max User count is reached. During each constant, users are executing scenarios constantly. Whenever users finish a scenario or encounter an error, they will immediately start another scenario, loading the site at a constant level.

You could apply 1,000 users at the beginning and run them over the course of the full hour, but you will get data that only tells you about 1,000 users. Using the ramp methodology allows you to pinpoint when events occur.

Duration	Max Users	Type
2	100	RAMP
4	100	CONSTANT
2	200	RAMP
4	200	CONSTANT
2	300	RAMP
4	300	CONSTANT
2	400	RAMP
4	400	CONSTANT
2	500	RAMP
4	500	CONSTANT
2	600	RAMP
4	600	CONSTANT
2	700	RAMP
4	700	CONSTANT
2	800	RAMP
4	800	CONSTANT
2	900	RAMP
4	900	CONSTANT
2	1000	RAMP
4	1000	CONSTANT

## Where do users spend the most time on your site?

The most frequently visited parts of your site are the most important to test. Focusing on them allows you to keep your scripts small and targeted, honing in on key processes or pages for very detailed data on problems.

Let's take a basic ecommerce site as an example. Users come to the homepage, browse, select some products, checkout and leave, right? Actually, some users come to the homepage and then leave, some browse around and then leave, some add items to their cart and then abandon the cart and leave. Some actually buy the product they added to the cart. So, you could write a large script that comes to the homepage, navigates every category, picks some products and checks out. However, that's not really how your site is loaded.

For this example you would want to create 4 test scripts as follows:

1. User comes to the homepage and then exits
2. User browses a category and then exits
3. User browses for a product and adds that product to the cart and then exits
4. User browses the site for a product and adds the product to the cart and then checks out

Once you create the scripts, you apply a percent of the load to each script based on real site traffic. Your checkout process may not be able to handle 2,000 concurrent users, but you may never have more than 5% of your users checking out. Meaning, the checkout function would need to handle only 100 concurrent users.

## How long do users take when navigating your site?

The next thing to consider is how fast you want the scripts to execute. By using computers we have removed the human element of looking over the page and figuring out where to click next. We have scripted this and can click the next link the moment a page has loaded. To better approximate reality you should consider adding user "think time" to scripts. If you have the right analytics data available, you can figure out how long a user spends on the home page before making a selection. As a rule, we use a function that randomizes the think time between 3 and 7 seconds after every page load.

## Do you use third party vendors?

Make sure your vendors are aware of your testing dates and times. If they're not notified, they may well see the spike in activity as an attack and blacklist the originating IP's. This would cause your load test to record either errors, as items wouldn't be available for download, or you could get invalid page load time data (not all the components of your page would be downloading for the test).

## What Makes a Good Script?

Simply put, a good load testing script simulates user behavior in the most realistic and accurate way possible. Before you begin scripting make sure you've outlined your scenario, identified a reasonable range of pause times between steps, and determined if your script will involve randomization or parameterization. Once you have the answers to these questions you're ready to get started.

### Tip

It's a good practice to include detailed comments and descriptions wherever possible on your script. They simplify your life and make a good reference for later use.

While best practices for scripting will largely depend on the scripting language and platform you're using, below are some tips for what makes a good script:

### **Content Validation Combined with Proper Use of 'WaitForPageToLoad'**

Content validation is the only way to confirm the right page got downloaded or the right action was performed. You can do this by searching for an appropriate piece of content in every step that confirms the validity of each downloaded page.

### **Setting Proper Timeouts**

Most WaitFor commands correspond to the default timeout. We recommend setting the timeout value before the actual transaction begins in the script. Studies show that the ideal page load time is 5 seconds. Anything longer doesn't work for real users. However, 5 seconds is too small of a timeout threshold for load testing, since load times will likely rise with increase in overall load. Hence, 60 seconds to 120 seconds is ideal.

### **Optimization of Locators**

By optimizing locators you'll create a more robust and reusable script. The most commonly used locator is the xpath. To find the correct xpath, consider using Firefox plug-ins like xpath finder or firebug.

### **Pause Times**

Inserting pause times in between steps enables you to simulate real user actions more closely. A range of pause times is recommended over a fixed time, since different users browse through a web site at different paces.

### **Randomization**

It's smart to have large datasets available for testing. Many bugs don't surface unless you test with thousands of users or mail server mailboxes. For scenarios such as product checkout or browsing product pages, consider randomizing the product from a list of several. This lets you simulate user behavior more realistically. After all, not all users are going to purchase the same product at the same time.

## **Test Execution**

Now let's consider the type of metrics your test is likely to harvest, plus ways to prepare for execution.

### **Execution checklist**

Before your load test starts, confirm that your script is still performing as expected. Ensure you're ready to capture CPU, Memory, Disk I/O and network throughput (plus any other useful stats) at a frequency of 2 – 4 times a minute. This should allow for fairly comprehensive averages throughout the test without taxing your system too much. Monitoring network devices such as routers, firewalls and load balancers will help you identify any network-related problems.

You'll also want to capture application logs such as Apache's error log. Any logging you do during a load test can hinder performance slightly, but will ultimately help you pinpoint more bottlenecks and errors. The data will help you fine tune the application, plus determine the additional hardware needed to handle the target load level.

While the test is running, it's a good idea to open a conference bridge with all responsible parties—system administrators, database administrators and developers. Having everyone's eyes on the system and application

during testing lets you diagnose issues in real time. It also helps you agree on a plan for resolving pressing issues. If you attempt a fix while the load test is running, consider pausing the test. If the fix is time-consuming and the test has ceased yielding useful data, consider stopping the test and running another later.

The most useful metrics while running a load test are: Number of successful transactions, number of page views, page load times, number and types of errors, and throughput. These are detailed further below.

### Successful transactions

What kind of real-time metrics will you get when the test is running? For starters, look for successful transactions. This shows how many users made it through the entire script flow without any errors. Bear in mind there will probably be a slight lag as reporting unfolds. With most testing tools, “real-time” is an approximation.

### Page views

Viewing the number of page views per minute the application is serving measures your total capacity and provides a simple benchmark. Utilizing a success/failure graph over the duration of the test will let you easily see a ratio of errors versus successes, plus whether the application was able to scale as users increased.

### Page load times

Page load times are vital in gauging the end-user experience. Generally, we’ve found that anything beyond 5 seconds is unacceptable, with drop rates rising dramatically as the wait drags on. For an e-commerce site, this is crucial—your competitors are just a click away.

Metrics that show the total or actual page load time will tend to report higher times. Thus, it’s important to distinguish the reported page load time from the perceived time. Full-page breakdown graphs can help pinpoint this difference and make it easy to see the impact on end users.

### Errors

When running tests with real browsers, you can see screen shots or video playbacks of real errors users encounter. To find what’s causing errors, it’s sometimes useful to view the HTML source or the request/response headers.

### Throughput

Throughput is a helpful metric to glance at from time to time. As you increase the amount of load on the system, the throughput should increase as well. If, as additional users are applied, throughput levels off and page load times increase, the culprit could be bandwidth saturation or over-utilization of servers. You can compare the maximum throughput provided by your Internet Service Provider (ISP) to the maximum throughput the test achieved. If the number is roughly the same, additional bandwidth may be needed or you may want to off-load certain page assets to a Content Delivery Network (CDN).

## Analysis

Once your load test is done, you'll want to make sense of it all. You'll especially want to point your developers and system administrators to the likely causes of problems you've uncovered.

### High-level results

To get an overview of your test results, begin with successful page loads per minute. This is a very high-level view, but it can help you identify if and when your site started having problems—and whether those problems were related to site performance or full-fledged errors.

Let's assume that we're looking at data for a one-hour test split up into 10 six-minute intervals. Also assume we applied 10% of the load during each interval, as we recommended for configuration. If you're lucky and your site can handle all the traffic you've thrown at it, you'll see results that look more or less like this:

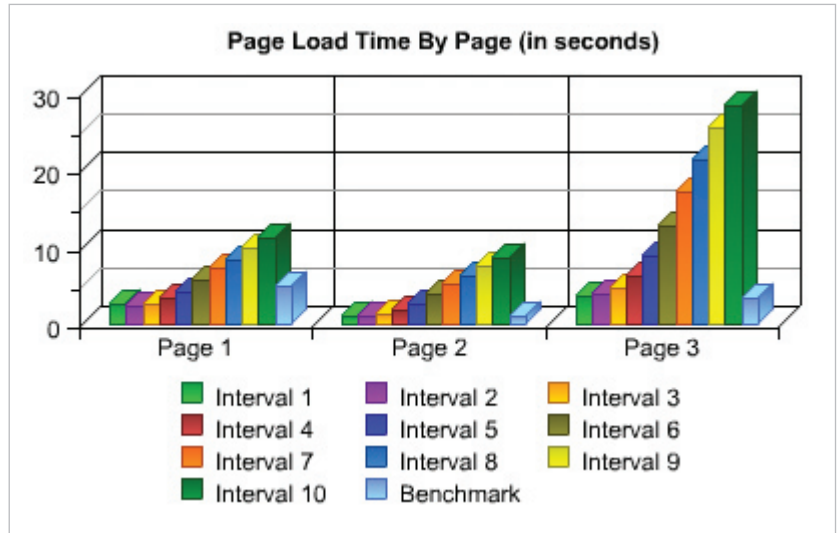


This chart shows the site started off serving 4,000 pages successfully per minute. Every six minutes it ramped up to serve about 4,000 more pages per minute. Since these ramp-ups correspond to increases in user load, we can see the site suffered almost no performance degradation during the test. This stair-step pattern is the ideal result for a test configured this way.

Observe, however, the stair-step blurs a bit at the end of the test. The last two or three steps aren't quite as steep as the previous ones and are differently shaped. This indicates that user load has begun to impact site performance. If the steps start to get shorter, you know the server is slowing down.

### Digging deeper: load times

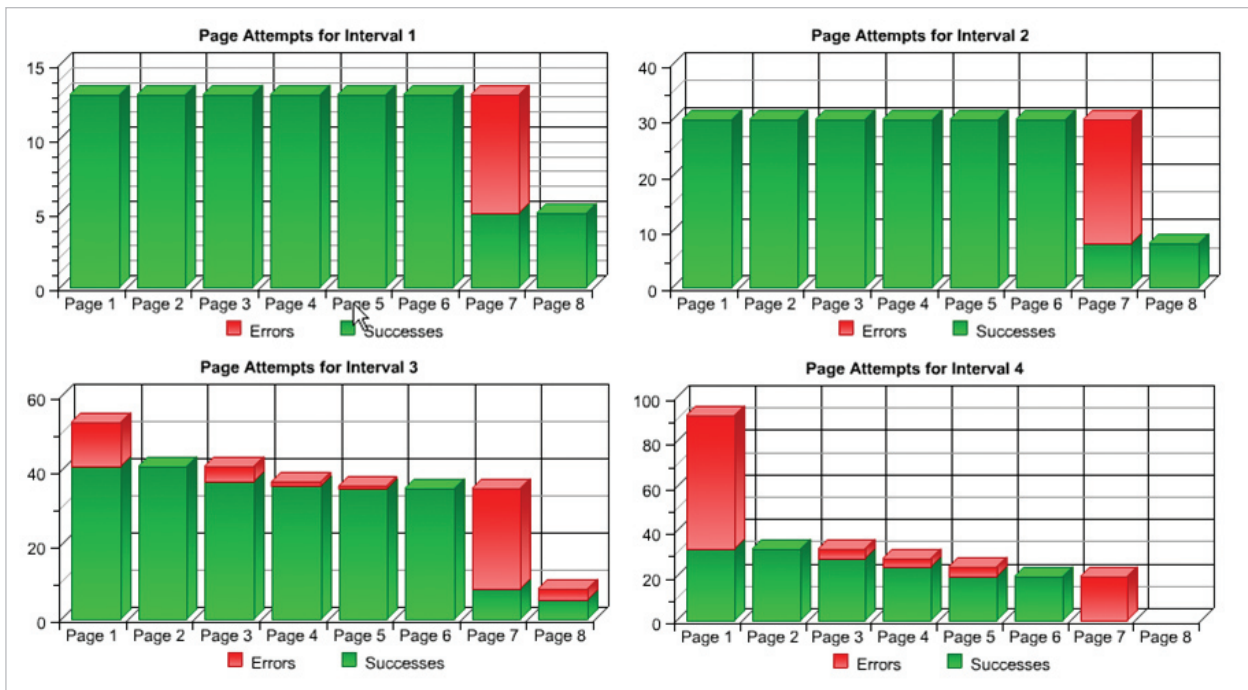
Now that we've seen our site is in fact suffering under the load, let's get a little more detail on where and when that happens. First, we need to identify which of our user scripts are having trouble. Examine the successes/failures graph for each individual scenario and use the same technique as above to find problems. Once we've identified which scenario is slowing down, we'll take a look at a graph of load times by page (see chart to right):



This graph shows average load time by page for each of the 10 intervals during the test. Within each page we can see the load time slow as more users are added, but page three looks particularly troublesome. The other pages load about four times more slowly at the end of the test vs. the beginning, but page three is loading almost eight times more slowly. In this case, we'll definitely want to take a deeper look to identify the problem.

### Digging deeper: errors

Finding errors might seem worse than discovering slow page loads. While that's true after your site goes live, there's no need to panic now. Errors are frequently easier to track down and fix than performance problems. Why? They leave more evidence. If one of your scenarios has errors, simply look at a chart of errors by page. It helps to group them by interval, so you can see how the results change as load increases.



These charts show the total number of page requests per page during each interval (color-coded to show successful and unsuccessful requests). This gives you an idea of how error distribution evolved over time. It's best to focus more on things that went wrong early in the test first. Often, early problems take up a large share of your server's resources, which can trigger errors on subsequent pages. **Important: You want to track down the root cause of the problem and not waste time looking at the symptoms.**

### Tracking down errors

Once you know where most of your errors are occurring, you'll want to find out exactly what's going on. Errors fall into two major categories:

- **Timeout Errors** – These are generally caused when you're seeing more network traffic than your infrastructure can handle, affecting the network or application server.
- **Content Errors** – A content error occurs when the website returns a page to the user, but it's not the page the user expected. These errors are normally the result of a configuration issue or a bug in the application that only appears under load. Examples: database row locking and other kinds of resource contention.

The error type can help point you in the right direction and tell you which of your server log files to start hunting through.

### Examples

Here's a list of a few things that might go wrong in your application and how they might manifest, both in your test results and your log files:

- **Running out of memory for sessions** – The application works fine at the beginning of the test, but suddenly collapses after a certain number of users. This will show up in your memory usage on the application server.
- **DB Connection Pool sizing** – The application continues to serve a fixed number of pages successfully despite a climbing error count. Most page requests end up failing to get a connection to the database or time out, both of which should show up as errors in your application log.
- **Slow SQL Query** – This usually manifests as a "high time to first packet" for the page, but should show up in the DB log as well. The database log will contain a record of how long each query took to execute. Depending on your application and the specific queries, you might find some queries perform much worse under heavy load than during functional testing.
- **Too many HTTP requests per page** – The first rule of building a fast site is to minimize the number of round trips to the server. This problem will show in your test results as a slow page; look for a waterfall graph with lots of rows. There are a number of tools that can help, such as Google page speed and YSlow.
- **Race condition in the application** – These errors tend to be a bit more transient depending on the current server load. Successful page loads and errors per minute might spike up and down unpredictably. You should probably identify a timestamp where the errors peaked all at once and check in the log files for errors near that time.

### Other Considerations

One more thing to keep in mind – you can always drill deeper. All of the charts and graphs we've looked at so far represent aggregate data. Unless you're looking at the individual rows in your test results, every data point you see has even more data underneath.

### In Closing

We hope this white paper is a useful guide to building an effective test plan, one that improves your web applications and performance initiatives. For more information on load testing, see our Webmetrics Performance Blog at <http://blog.webmetrics.com>. You'll find additional tips on planning, configuring, scripting, executing and analyzing your tests. By doing your homework, you'll breathe easier on launch day—and every other day.

---

### About Neustar® Webmetrics® Load Testing Services

When companies launch or upgrade websites, they need to know heavy traffic won't result in crashes. The same goes for ecommerce sites gearing up for peak selling times. Neustar Webmetrics Load Testing services help businesses know they're ready. By testing loads they can identify a website's breaking point, find service bottlenecks and accelerate smart fixes. To ensure the most accurate views of what real customers experience, Webmetrics conducts load tests with real browsers. We also offer both on-demand service through our BrowserMob brand and full-service testing with a dedicated engineer. By helping avoid unpleasant surprises, Neustar Webmetrics Load Testing helps sustain your growth while protecting your brand.

**For more information, please call: +1-888-367-4812**

**Or, come check us out at [Webmetrics.com](http://Webmetrics.com) and [BrowserMob.com](http://BrowserMob.com)**