PerfTestPlus

Better Testing... Better Results

Right Click → View Source
And other tips for performance testing the front end

Right Click

→

View Source

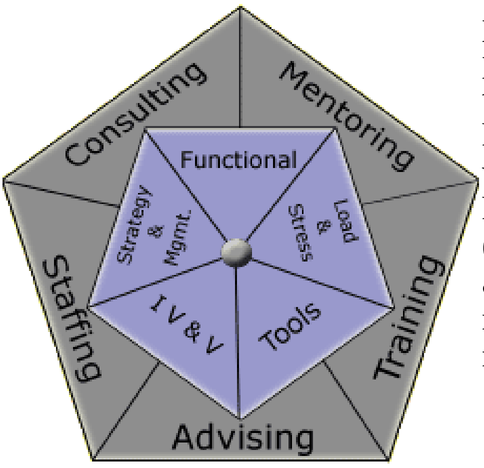
by:

R. Scott Barber

I recently read *High Performance Web Sites: Essential Knowledge for Frontend Engineers* by Steve Souders, O'Reilly, 2007. The book is sub-titled "14 Steps to Faster-Loading Web Sites." Before you stop reading because this is a book written for developers, consider the following:

- The research Souders presents suggests that approximately 80-90% of a web page's response time results from front-end design decisions. My experience suggests numbers more like 50-80%, but most of my experience comes from projects where existing multi-user applications are being retro-fitted with a web-based front end and/or applications with significant back-end performance issues that I have been called in to help find.
- Virtually all of the tools, training, articles, and conference talks available to individuals who test the performance of software systems are heavily focused on the back-end. So much so, in fact, that most dismiss the front-end aspects of system performance as something not worth worrying about, ostensibly because we can't control the end-user's system.
- In my experience, the front-end design and development of web sites is conducted with little to no thought to performance, outside of possibly reducing the size of the graphics. Additionally, I've never been made aware of a team conducting peer review on the HTML that generates the web page on the client side, never been made aware of unit tests on such code, nor witnessed a tester deliberately and proactively testing the HTML for possible performance issues.

Put these things together, and what you get is no one paying attention to, or checking for, potential performance improvements in the part of the web page most likely to contain the most opportunities for the largest and cheapest performance improvements. Reading this book, I realized that I frequently test for most of these front-end performance issues without realizing it, that he mentioned some I likely would have never thought to test for, that there were a few front-end performance issues mentioned in the book I wouldn't have even known to test for, that it's been a mistake to not test for these items (or call them out while teaching testers) more deliberately, and that these tests are low-cost, both in terms of time and in terms of the tools and resources required. In fact, I frequently conduct most of these tests during the first 15 minutes of performance testing I conduct on a web site, though I admit that in 15 minutes I can generally only test a couple of pages.



Throughout this article, I describe how to conduct tests manually, using load generation tools, network protocol analyzers, helper websites, and browser plug-ins. I have validated the techniques using the following free tools – not free trials, but free-with-no-strings-attached. If you work for a company that doesn't permit the use of free tools, I'm certain that a quick web search will help you turn up dozens of alternatives that will cost your organization enough money for them to take the tool seriously (for those of you who think this is a joke, it's not. More than 50% of the teams I consult with and individuals I train report not being permitted to use freeware, open source software, shareware, or even time-limited free trials on company machines or networks.).

- Free or Open Source Load Generators (a.k.a. Performance Testing Tools)
  - JMeter (http://jakarta.apache.org/jmeter/)
  - WebLoad (http://www.webload.org/)
  - OpenSTA (http://www.opensta.org/)
- Free or Open Source Network Protocol Analyzer
  - Ethereal (http://www.ethereal.com/)
  - Fiddler (http://www.fiddlertool.com/)
- Free Browser Plug-Ins
  - Firebug (http://www.getfirebug.com/) with YSlow (http://developer.yahoo.com/yslow/) for Firefox
  - HttpWatch (http://www.httpwatch.com) for IE
- Free Helper Websites
  - Web Page Analyzer - from Website Optimization: *Free Website Performance Tool and Web Page Speed Analysis* (http://www.websiteoptimization.com/services/analyze/)
  - Gomez Instant Site Test (http://www.gomez.com/info_center/instant_test.php)

With that, let's take a look at some of the tests you can conduct with no more training than you'll get in this article, tests that could lead to dramatic improvement in end-user response times without requiring that you look any deeper than the web server.

Number of HTTP Requests

Web pages are not retrieved via a single transaction. Pages generally include a single request for the HTML document, one or more requests for stylesheets, one or more requests for external scripts, and multiple requests for graphics, multi-media content, and third party content such as advertisements. Even when many of these objects are stored locally in the browser's cache, a request is still frequently sent to the server to determine if the object in the cache is still "fresh." What this means is that each object used in rendering a web page carries with it significant potential for increased overhead, and thus degraded performance from an end-user perspective, even when the client has a "primed" browser cache containing the object in question. Determining the number of requests a page makes, and what it is requesting, can be done in several ways.

No matter what method you use, you will want to begin by either clearing your browser cache or requesting the page twice (one time using ctrl -> refresh to override the browser cache) to ensure that you can view all of the requests. Since the following methods only collect actual requests, notclearing or overriding the browser's cache could cause an incomplete collection and lead you to think that no stylesheets, scripts, images, or multimedia content is being requested, depending on a variety of settings and conditions you may or may not be aware of and/or have any control over.

1. If you have access to a load generation tool or a network protocol analyzer, you can simply start recording, then navigate to the page or pages of interest. Search your recording for "GET" statements, and make a note of what objects are being requested. Remember that, with some tools, the default view of the recorded script may only contain the base HTML request, not the child requests (i.e. requests for linked stylesheets, external scripts, graphics, etc.), thus requiring an additional step to view all of the requests.
2. If you are testing in an environment where you are permitted to install browser plug-ins, you have several options available to make this task simple. The plug-ins I recommend, depending on which browser you use or wish to test against, are:
    a. Firebug with YSlow for Firefox.
    b. HttpWatch for IE.
3. If you have no tools available, you still have several options:
    a. Visit one of the helper web sites listed above, type the URL for the page you want to test into the text box, and push the "Submit" button.
    b. In Firefox, right click on the page, select Page Info, then navigate to the Media tab. Note that this method does not reveal scripts and stylesheets, but will still show you requested graphics and other multi-media content.
    c. In IE, right click on the page and select View Source. In this case, you will need to search the code for "link" and "img" tags. Additionally, if you find links to stylesheets (any link to a .css file), you will also need to download each stylesheet by manually requesting it from the navigation bar and searching it for "url" entries which may be used to request scripts, images, or multimedia content. (This is by far the most cumbersome method, but it will still get you the information).

Armed with your list of requests, the first thing you are looking for is volume—excessive requests slow things down. There are several indicators to look out for to decide whether or not the requests you see are excessive.

1. **More than one request for an external stylesheet.** While it is sometimes a good design decision to separate page styles into more than one external stylesheet, this is not common and is certainly not generally helpful in terms of performance. Generally speaking, maintaining more than one external style sheet is a good idea only if there is one base stylesheet that applies to many web pages and a second, large, stylesheet with styles that only apply to a few web pages.
2. **More than one request for scripts from the same domain.** Even though there are more sound reasons for linking to multiple external scripts than linking to multiple external

stylesheets, it still not common for links to multiple external scripts to result in better performance than linking to a single, consolidated external script. For example, if the page you are testing is a complex data entry form, it may make sense for the form validation script to be separate from other scripts that are common to all pages. Doing so would reduce the size of all pages except the form, thus improving performance on those pages, even though the extra request would degrade performance slightly on the form page. Regardless, more than one external script is at least worth asking about.

3. **A lot of graphics.** I can't tell you how many is "a lot," but I can tell you that currently, IE7 and FireFox 2.x default to 2 parallel downloads per hostname (for HTTP/1.1 web pages, which most new sites are). This means that, no matter the size of your images, the browser will only download 2 at a time, and that the next two won't start until **both** of the preceding two are complete. This is different from HTTP/1.0, where FireFox defaulted to 8 parallel downloads per hostname and IE varied by version but never to fewer than 2. The impact of this change is that using the fewest graphics of approximately the same size tends to result in the best performance. This is counter to the notion that smaller graphics are always better. Combining several small graphics into one or two larger graphics frequently improves performance. There is, of course, a point of diminishing returns. Graphic size vs. number of graphics is something that is worth working closely with the front-end designer on, to test various options in search of optimal performance. A well respected "rule-of-thumb" to guide your decision-making in these matters urges caution if a page contains more than 12 total requests. This number is used by Souders in his book, Andrew B. King in *Speed Up Your Site: Web Site Optimization*, New Riders, 2003 and most convincingly by Aaron Hopkins in an article published on his site titled "Optimizing Page Load Time" (http://www.die.net/musings/page_load_time/).

Sequence of HTTP Requests

Many of the same methods can be used to determine the sequence of the objects being requested when a page loads. The exceptions are the helper websites we discussed previously and Firefox's "Page Info" screen, as these group and/or sort requests by type and size to highlight volume and size issues rather than sequence issues. The sequences you are most interested in are:

1. **Request stylesheets first.** Web pages will either not display at all until stylesheets (.css) have been downloaded or appear to refresh themselves once the stylesheet is retrieved. For this reason, it is critical that stylesheets are among the first items requested following the base HTML page.

2. **Request scripts last (or at least late).** Once a script is requested, no other objects will be requested until the script has been completely received. Additionally, browsers cease displaying content while scripts are downloading. This means that any objects that complete their download after the script has been requested won't be displayed until the script has been completely downloaded, thus making the rendering of a web page appear to stall. This means that it's highly desirable for scripts to be requested after the objects that are most interesting to the end-user. Remember, most end-users perceive responsiveness based on the time it takes for the content they are interested in to appear, rather than the time it takes the entire page to load.

Whether you are viewing the HTML source or captured requests, what you are looking for is that stylesheets are requested first and scripts are requested either last or at least very near last. There is a common argument that scripts controlling user interactions such as image maps and roll-over objects should be requested early so that the user will have the “proper” experience, even while the page is downloading. In my experience, however, a user is much more likely to become frustrated or abandon a website if it appears to be stalled while downloading than if a roll-over image or image map isn’t enabled until the page downloads completely.

Redirection and/or Hidden Errors

You can use the same methods you used to check for appropriate request sequencing to check for redirection (3xx series response codes), client errors (4xx series response codes), and server errors (5xx series response codes). In this case, the main indicators you are looking for are the following:

1. **Excessive 3xx series response codes.** 3xx series codes indicate that the request was processed, but that the browser must retrieve the object from another location – resulting in additional request/response pairs. While there are plenty of sound reasons for the redirection of some requests, it’s worth making sure that redirection is being done intentionally and for good reason. For example, redirecting from a removed web page to its replacement, or redirecting from an obvious misspelling of a web page to the correct page is a good reason. Redirecting requests for images because the image’s parent directory has been moved but no one has bothered to update the link is probably not a good reason accept the inherent performance degradation associated with the additional requests associated with redirection.
2. **Any 4xx series response codes.** A 4xx series code is returned when there is a problem with the client request. The most common is 404, which indicates that the requested object was not found on the server. Generally speaking, if the web page is displaying and functioning properly, but individual requests are returning 4xx codes, that indicates that the page is simply requesting unneeded objects and taking extra time to do so.
3. **Any 5xx series response codes.** 5xx series codes indicate that an error occurred on the web server while trying to fill the request. Any 5xx series code should be of interest to the development team.

I refer to these as hidden errors because when they occur for objects other than the base HTML document they are frequently not obvious or even visible to the end-user. Sometimes seeing these response codes is also indicative of deeper errors, but, they all result in requests that do not contribute to the display or content of the page and are frequently entirely unnecessary.

HTTP Response Headers

To check HTTP response headers, you will need to use a load generation tool, network analyzer, or one of the browser plug-ins. If you don’t have any of those tools available, another helper web site might be of value. I suggest Peter Forret’s "View and analyze HTTP headers" page (http://web.forret.com/tools/analyze.aspx), where you can enter the URL of a web page and the site will retrieve a list of the HTTP headers sent back by the web server, so you can check page expiration and caching settings. The details about what parts of the response are appropriate vs. unnecessarily performance-inefficient are highly dependent on variables such as the frequency with which the site and/or objects change, the frequency with which users of your site visit the site, and the relative risk of those users viewing stale content. Nonetheless, the following items are consistently worth inspecting:

1. **Check for an appropriate Expires: entry.** If the HTTP response for an object does not include an Expires: line, every time a user requests a page containing that object, a request will be sent to the server to determine whether or not the cached version is "fresh." If you have objects that are unlikely to change frequently (for instance, the company logo) you can avoid the "freshness check" request with a date/time in the Expires: line that is far in the future. Expires headers are most often used with images, but they are often also appropriate for other components including scripts, stylesheets, AJAX, and Flash components. Look for objects with no Expires: line and for Expires: entries that seem inappropriate to you.
2. **Check for appropriate ETags.** Entity tags (ETags) are a method of identification that web servers and browsers use to determine whether or not the object cached on the client's machine matches the one on the server. The challenge with ETags is that they are generally unique to a specific web server, meaning that using them may actually be detrimental if the web site has multiple web servers. If you know that the web site uses a single web server, ETags are probably a good idea. If the web site uses multiple servers, you will want to inquire about whether the multiple servers have been accounted for, or recommend that the ETags be removed.
3. **Check other cache controls.** You may or may not observe other entries following lines such as Cache-Control:, Last-Modified:, Pragma:, Set-Cookie:, and Age:. If you do observe those lines, ensure that the entries make sense to you. If you don't observe those lines and feel like they should be there, bring it up to someone.

The bottom line is that you want to check HTTP response headers to determine whether or not the web site has been configured appropriately to take advantage of browser caching on the client side. Frequently, the only way to determine the appropriateness of these entries is to spend time with administrators and architects discussing both how the site is used and how it has been designed, specifically related to client browser caching.

Source Code and Objects

Finally, if you haven't done so already, you'll need to manually examine the source of the HTML, .css, scripts, graphics, and other remaining objects. To date, I have not found any specific tools that save time over manual inspection in enough situations to recommend for these final front-end performance testing tasks, although HTML, script, and graphics editors appropriate to the web site are generally useful. The final front-end performance testing tasks that I recommend are:

1. **Ensure that HTML source code does not included embedded scripts and CSS expressions.** It is extremely rare that performance is improved by including scripts and CSS elements or expressions directly in the HTML. The reason for this is simple: the base HTML for a web page is the part of the page that is most frequently updated and therefore least frequently served from cache. Since the HTML is so much more likely to be downloaded every time, it only makes sense to keep it as small as is reasonable. Keeping scripts and CSS elements external to

the HTML, and thus cacheable, is virtually certain to improve performance, on average, over time, across the users of the web site.

2. **Ensure that styles and scripts are not duplicated.** In my experience, stylesheets and script files are notorious for containing duplicate or overlapping content. Sometimes content is duplicated across separate files; other times it is duplicated within the same file. While you may not want to spend the time to do a complete review, a quick scan of the source can often reveal whether or not there is significant overlap or duplication.
3. **Check for code minification.** Believe it or not, "minification," at least according to the Random House Unabridged Dictionary, is a valid English word meaning "the act of minimizing." With regard to computer code, it refers to condensing and optimizing the code to perform the desired function using the fewest lines and/or characters of code. While inspecting the HTML source code, external script files, and stylesheets, you want to look for excessive comments, white space, line breaks, variable name length, and other items that increase file size.
4. **Check the appropriateness of graphics' size and compression.** It may seem obvious, but many web designers are still using graphics in formats that have unnecessarily large file sizes, in sizes different than the height/width they are to be displayed in, and of a quality well in excess of what is necessary or reasonable for the purpose of the web site they are being displayed on. In general, .gif formatted images compressed to 64 or fewer colors are more than adequate for most graphics and thumbnails; .jpg formatted images compressed to 256 or fewer colors are typically adequate for photographs; and it is rarely justifiable to use HTML height/width properties to shrink or stretch an image rather than creating a new image of the correct size.

In each of these cases, use common sense as your guide. For example, some web sites reduce all of their file, directory, and variable names to two or fewer characters each as a matter of policy to minimize file size. From a purely performative perspective, this is excellent; however, the additional work required to document and/or maintain the code makes this practice completely unreasonable for most web development efforts. You will have to work with your team to find the proper balance between duplication/ minification/ compression and practicality.

Summary

This article describes several tests that can be used to determine if a web site is likely to exhibit poor front-end performance. Identifying these areas of potential performance improvement could result in a 50% or greater reduction in the user-perceived response time of the web site. I am confident that once you get your tool box of applications, plug-ins, and helper web sites in place, and practice these tests just a few times, you will be able to scan a website for significant offenders of each of these items in less time than you just spent reading this article. With such a significant potential for dramatic performance improvement, and such a small investment in time and tools required, I see absolutely no reason why any web site should go live without these tests being conducted.

About the Author

Scott Barber: Tester, Author, Speaker, Disrupter & Dad

A prominent thought-leader in the area of software system performance and testing software systems in general, Scott Barber, founder and Chief Technologist of PerfTestPlus, makes his living writing, speaking, consulting, and coaching with the goal of advancing the understanding and practice of software testing. Scott has contributing to four books (co-author, *Performance Testing Guidance for Web Applications*, Microsoft Press; 2007, contributing author *Beautiful Testing*, O'Reilly Media; 2009, contributing author *How to Reduce the Cost of Testing*, CRC Press; 2011, author *Web Load Testing for Dummies*, John Wiley & Sons, Inc.; 2011), composed over 100 articles and papers, delivered keynote addresses on five continents, served the testing community as the Executive Director of the Association for Software Testing, and co-founded the Workshop on Performance and Reliability.

Today, Scott is focused on applying and enhancing his thoughts on delivering world-class system performance in complex business and technical environments with a variety of clients and is actively building the foundation for his next project: driving the integration of testing commercial software systems with the core objectives of the businesses funding the creation of those systems.

When he's not "being a geek", as he says, Scott enjoys spending time with his partner Dawn, and his sons Nicholas and Taylor at home in central Florida and in other interesting places that his accumulated frequent flier miles enable them to explore.