Web Page Size, Speed, and Performance

Terrence Dorsey

“Velocity is the most valuable conference I have ever brought my team to. For every person I took this year, I now have three who want to go next year.”

—Chris King, VP Operations, SpringCM

Join business technology leaders, engineers, product managers, system administrators, and developers at the O’Reilly Velocity Conference. You’ll learn from the experts—and each other—about the strategies, tools, and technologies that are building and supporting successful, real-time businesses.

O'REILLY®

Velocity

CONFERENCE

BUILDING A FAST & RESILIENT BUSINESS

Santa Clara, CA
May 27–29, 2015

http://oreil.ly/SC15

©2015 O’Reilly Media, Inc. The O’Reilly logo is a registered trademark of O’Reilly Media, Inc. #15306

Web Page Size, Speed, and Performance

Terrence Dorsey

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Web Page Size, Speed, and Performance

by Terrence Dorsey

Copyright © 2014 O'Reilly Media, Inc.. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Mike Loukides and Brian Anderson
Production Editor: Nicole Shelby
Cover Designer: Randy Comer
Interior Designer: David Futato
Illustrator: Rebecca Demarest

June 2014: First Edition

Revision History for the First Edition:

2014-06-03:	First release
2015-03-24:	Second release

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *Web Page Size, Speed, and Performance* and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-1-491-95022-7

[LSI]

Table of Contents

Introduction

The performance of your website is as valuable as ever, and grows in importance every year as ecommerce takes a bigger bite of the sales pie, mobile online use continues to grow, and the web in general becomes ever more entangled in our lives. There's one significant problem: websites seem to be getting slower every year, not faster.

There are many reasons why websites are slow. Servers are an important aspect of web performance, and in some respects that's a solved problem, but there are still many high-profile sites that demonstrate poor performance even with sophisticated hardware. If you're serious about it at all, you're not going to be running a site on shared hosting and expecting to handle significant traffic surges without a problem. Building out a site with load balancing, dedicated database resources, and so on is simple enough. It's really a matter of a little expertise, some planning, and a budget. But adding server resources before fixing more basic performance issues brings its own problems.

Web pages themselves present a different issue: they keep getting bigger and more complicated. Radware and Strangeloop have been measuring web page size and performance for the Alexa top 2,000 retail websites going back to 2010. Their most recent report noted that, in 2014, web pages among the sites examined are bigger than they've ever been. The average web page for the top 500 Alexa sites was a whopping 1.4MB. Yes, megabytes!

Make the Web Faster

Moore's Law takes care of this, right? Faster processors, bigger pipes, better software. Applications used to come on a single 1.44MB floppy.

Now they require a DVD (or more likely a several-gigabyte download), but they run just as fast while providing more features. This means big web pages shouldn't be a problem, right? Just throw more technology at your site—faster servers, more RAM, caching and load balancing services, a NoSQL database—and the problem should be solved.

Unfortunately, it's more complicated than that. Your backend hardware and software are only one facet of the performance problem. There's certainly an opportunity to performance-tune the backend of your site either by optimizing its configuration or simply adding more horsepower. However, along this path you're likely to encounter the need for much deeper technical knowledge along with added costs and complexity.

There's another path that I think you should explore first. Plus-size web pages are a problem all on their own. A large web page takes up a great deal more bandwidth in transit and many more round trips for everything to arrive on the computers of your visitors. The more visitors, the more bits flying around. Your first task should be a simple one: examine whether the size and complexity of your web pages themselves are creating a performance bottleneck.

The Simple Path to Performance

Making your web experience quicker and more enjoyable for visitors may not require a large investment in new development languages and frameworks or state-of-the-art hardware. It may be as simple as making sure the basic HTML, CSS, and JavaScript components of your site —the foundational frontend building blocks of the Web for nearly 20 years—have been optimized. This is all well-known technology and *shouldn't* be a problem, but evidence shows us it *is* a problem… and it's getting worse from year to year.

In this article I'll explain why web pages have become so large, and I'll take a look at why that's a problem for visitors to your website — and possibly a problem for your business. Then I'll show you a few simple but often overlooked frontend development techniques to help whip your web pages into shape, slimming them down and resulting in the best performance possible.

Big Web Pages are a Bigger Problem Than You Think

Previously, I mentioned the Radware/Strangeloop reports on web page size and performance for the Alexa top 2,000 retail websites. The unbridled growth of websites revealed by these reports is pretty shocking and it has some unfortunate side effects in terms of web performance. Let's take a closer look.

Thanks to rapid development in web servers, browsers and the technology sphere in general, some of the measurements are difficult to correlate over the entire span of these reports. Looking at comparable statistics, however, we can see that average page size grew from around 780KB and 86 resources in 2011 to 1,100KB in 2012 and over 1,400KB and 99 resources by the time of the early 2014 State of the Union Winter Report (see Figure 2-1).

To dig a bit deeper into the component specifics of this website growth trend, I headed over to the HTTP Archive. Using the "trends" tool, I chose to display data for the top 1,000 sites from May 2011 to early 2014. This report illustrates the growth in total page transfer size and number of requests over that period, then breaks the stats down by component, including HTML, JavaScript, CSS, images, and more.

I'll just show the total and HTML trends here, but you can clearly see that average page size for these sites starts just under 700KB in 2011 and rockets up to over 1,400KB by early 2014. Every category except Flash transfers shows similar growth over this period, and Flash remains roughly level in transfer size while declining in popularity (see Figure 2-2).

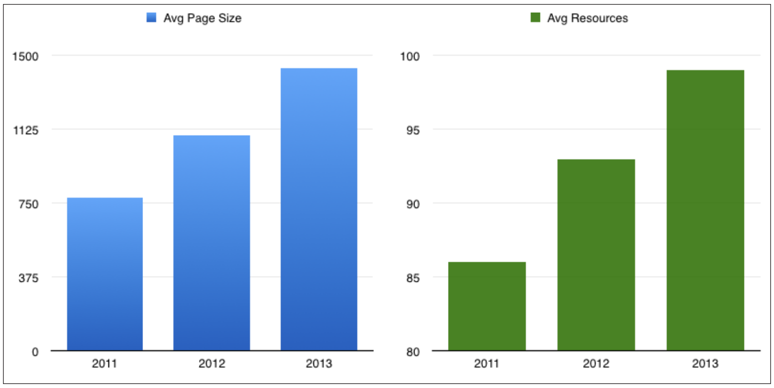


Figure 2-1. Growth of average page size and resource requests since 2011

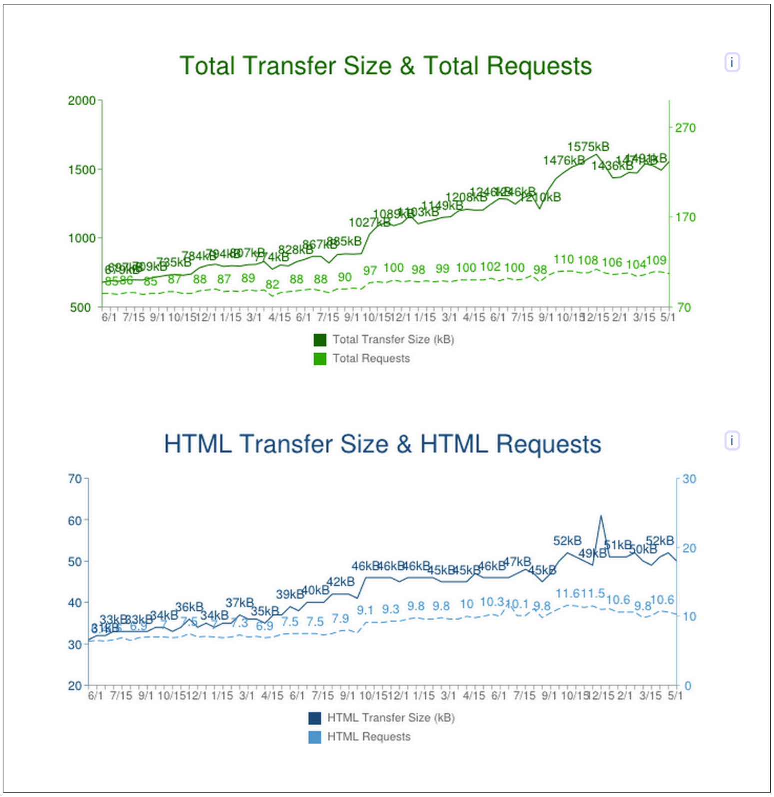


Figure 2-2. Trend data from the HTTP Archive

Looking for some additional comparable data, I went to the Internet Archive and pulled up versions of Amazon's home page from the April archives of each year starting in 2011. Using the Network tab of the Chrome developer tools, I measured the traffic created by loading each version of the site.

The archived 2011 site loaded 456KB in 81 requests; the 2012 site loaded 467KB in 68 requests; the 2013 site loaded 658KB in 126 requests; and the archived 2014 site loaded 777KB in 134 requests. That's a 70 percent increase in page size and a 65 percent increase in the number of requests needed to load the page. It's not a perfect test, but the results provide an interesting time-lapse comparison nonetheless.

For further comparison, I also loaded the current Amazon site: 2.6MB loaded with 252 requests!

Web Performance Means Business

Why is the performance of your website important? Why should you care about how big your web pages are becoming?

Here's one example: US ecommerce at the end of 2013 accounted for around 6 percent of all retail sales. Online retail spending is growing at an annualized rate of between 14 and 17 percent in the US, and the mobile portion of those sales has been increasing in double digits year over year as well. Outside the US, online sales are increasing even faster in areas like the UK, South Korea, and China.

The problem is performance. The rendering time for a site and the time to interaction (the point at which users can actually do something on your page) have a direct impact on customer experience. A good experience is the difference between converting a visit into a purchase (or some other desirable outcome) and having the visitor close the tab before your page has even rendered.

It's not just about ecommerce—the same rules apply to the frontend user experience (UX) of any website, regardless of whether you offer news, marketing, customer support, entertainment, personal blogs, or any other kind of content. But the same rules apply: if you want people to visit your site and stay long enough to have the first page render, that first UX impression is vitally important. Whether the site is a Fortune 500 retailer or Uncle Joe's fishing stories, the numbers tell us that people don't visit if the frontend experience is slow.

Charting Web Performance Against Business Success

Let's get back to those web performance survey results for a minute. Tammy Everts, discussing them at Web Performance Today, noted that the average rendering time for surveyed sites was over 5 seconds, one quarter of the pages reviewed took over 8 seconds and "5 seconds is still a far cry short of 3 seconds—the point at which the majority of shoppers say they'll abandon a page."

In another blog post, Everts notes that "A 2 second delay in load time during a transaction results in abandonment rates of up to 87 percent," a 24 percent increase over typical abandonment rates.

Leave Room for Social Experiences

According to Jainrain, the social media consulting company, almost 60 percent of shoppers do research online before buying, and half of the consumers who do that research then post comments or reviews about their purchases. As the report states, "71 percent of shoppers rely on customer reviews to influence purchase behavior." Plus, shoppers who log into an ecommerce site are more valuable, since they make purchases more often and are "nearly twice as likely [to] purchase on a site that automatically recognizes them."

That just scratches the surface of online behavior shaped by both performance and site features. You'll want to balance a svelte site that's quick to render and become interactive with more complicated features like comments, ratings, sharing and membership, or whatever else drives interaction between your business and its customers.

Whether you're selling online or promoting your latest open source project, the same rules apply. If you're not available to your audience, then you're needlessly turning them away.

Bigger Pages Clog Up the Pipes

The problems with heavyweight web pages start with the fact that, as you're browsing the Internet, it takes more packets to get large pages from the server to your computer. In some particularly wired countries or regions such as Hong Kong, South Korea, and Switzerland, there's plenty of bandwidth. And compared with streaming video from Amazon, Apple or Netflix, you're still talking relatively small amounts of data—you need at least 2Mb/s for sustained streaming video—even in in the United States where average home broadband bandwidth ranks only 33rd according to recent statistics from netindex.com.

But we are starting to see potential problems on the horizon. Not to pick on Comcast, but if their proposed merger with Time Warner's broadband operations is approved, the combined company will serve about 30 percent of US cable internet subscribers. Official statements about this merger touted "its ability to deliver groundbreaking products" along with "operating efficiencies and economies of scale," but no mention is made of investment in Internet infrastructure, improved quality, or faster service offerings.

Net Neutrality Affects You, Too

There are other industry issues that can affect the speed at which your web page traverses the network to customer browsers. The last-mile providers—Comcast and Verizon in particular—have a less-than-friendly relationship with internet backbone companies like Level 3 and Cogent. Contractual arguments over peering agreements have led to accusations of service interruptions and slowdowns, as well as complaints about a lack of investment in delivery infrastructure, leading to overburdened switches during high-use times of day.

More recently, we've seen Netflix and Apple strike payment agreements with Comcast to ensure network access levels, and comments from the FCC suggest deals like this might become policy, not anomaly. Clearly these companies anticipate a future where bandwidth could be a scarce resource, but they are profitable enough to pay for it for access (for the moment). If your web page needs more bandwidth, are you ready to pay up? And can you afford to compete with Apple for that bandwidth? It's becoming less of a conspiracy theory and more a reality of doing business every day. It also makes investing in development skills and practices to keep your own use of the Web light and efficient look potentially cheap in comparison.

What Makes Web Pages Fat?

Web pages don't add bytes all on their own—there must be something about development tools, programming techniques, and business practices that is making websites bigger, more complicated, and often slower.

The most obvious reason to make pages bigger? Because we can. Compared to the early days of the Web, server and client hardware performance is orders of magnitude greater. Bandwidth available to most users is far greater as well (on the desktop, at least). Even compared to the heady dot-com boom days over a decade ago, we've progressed far enough in web application programming tools and techniques that the focus is increasingly on polish and "experience."

We're also able to focus those tools and techniques on building polished experiences around specific business requirements. Done right, signing up for a service online, making a purchase, or chatting with friends (in a manner that gives advertisers a laser focus on your demographic) has never been easier or more enjoyable.

Weighed Down By User "Experience"?

Note, however, that I use scare quotes around experience. As I pointed out earlier, the experience of your website has a bottom-line impact on your business. Specifically, your business is affected by how quickly it renders and becomes interactive, as well as by the kinds of features you've chosen to implement.

Security is, of course, a concern—losing user data to hackers tends to turn consumers off—but registering and remembering returning users is a feature customers value. Again, ratings and reviews have a positive impact on sales, as do social referrals, one-click ordering, and online customer service. Just thinking about the business concerns on a typical ecommerce site, you can start to imagine the code piling up.

It may seem like my focus here is on ecommerce. If the focus of your site is something other than selling things, you might think these concerns are irrelevant. But think about it this way: whatever the purpose of your site, your concerns are, at some level, similar to those of ecommerce. You want people to visit your site, read what you have created, click links and share with their friends and colleagues. None of that happens when the site experience is slow and unresponsive.

Are Development Tools Part of the Problem?

An obvious question to ask is whether modern web development tools contribute to code bloat and performance bottlenecks. There are some seemingly obvious suspects in this crime.

At its heart, a web page just needs HTML and content. Let's assume you're keeping faith with the semantic markup concept, so your HTML markup represents just the objects in your content. You'll include a layer of CSS to specify the rendering details of your markup, with a little JavaScript sprinkled in to add client-side computation of dynamic content, AJAX interaction with the server, and more.

Generally speaking, this is a pretty simple game so far. It's text all the way down. But, as we'll see later, it is possible to drag down the performance of your site with just these seemingly straightforward ingredients.

I don't think the tools themselves are the problem. The building blocks aren't a problem either if we use them wisely. The problem lies, I think, in using the tools available to you unwisely, without understanding (or having forgotten) some of the lessons of the far slower, baud-limited past.

Let's take a look at a few of the basic elements and how they are used —and sometimes abused!

The Solution Starts with Understanding the Problem

As I noted earlier, the size and number of images used in web pages continues to increase, and there are a number of reasons for this trend. On the client side, a trend toward larger desktop computer screens means more real estate to fill—often with images. Once upon a time, a 600 pixel-wide page was considered lavish. Today, 1920×1200 pixels is a run-of-the-mill desktop, and high-density displays are pushing even greater pixel counts. All of this screen real estate tempts designers, long cramped by those small target screens, to fill the expanding canvas.

Hero Images and Scaling for Retina

The full-page-width "hero" image is one idea that's currently in vogue. Another that is getting tired, but won't go away thanks to readily available design templates, is the slide show. Three, four or even more high-quality stock images whisked across the internet and sliding across the screen every few seconds… and ignored while your customer searches for link to actual content.

High-density displays, led by Apple's Retina devices, probably aren't helping matters. Compared to traditional computer display pixel densities between 72 and 130 pixels per inch (PPI), high-res displays now use pixel densities of 220 PPI or more. (The current iPhone 5s and iPad Mini are 336 PPI. Some devices offer even higher resolutions.) This makes individual pixels mostly undetectable to the majority of users at standard viewing distances. The high-density displays look great on phones and tablets. Now they're on MacBooks, too, and probably coming to desktop monitors before you know it.

Web designers who are clued in to the issues around Retina want to make sure their sites look great on these displays. You can still use standard resolution images—by default, browsers will scale up images to display correctly on the screen. The problem is that upscaled images may look blocky and of poor quality.

Changing images to look good on high-density screens requires double pixel density images. These kinds of images do require some thinking ahead to create, but it's not rocket science.

Sizing Images Effectively

One technique involves creating images scaled to various supported pixel densities. For example, you might start with a 2× image at full resolution and downscale versions to 1.5× and 1× scales. (Start with the largest image and work down to smaller ones for best quality.) If the original image was, say, 200×200 pixels, you'd create additional versions at 150×150 and 100×100 pixels. Then you can employ user agent media queries and CSS for on-the-fly higher resolution image substitution when the display PPI supports it.

However, that's a lot of overhead for the designer and whomever keeps track of the code. Another technique is to simply use the 2× image everywhere and rely on the browser's down-scaling capabilities, which are universally very good these days, even in mobile browsers (and far better than up-scaling, since you're taking information out of the image, not trying to put more information in).

The problem is that these "Retinafied" images are not only twice as big as their regular-density siblings, they're using up to four times as many pixels.

Plug-Ins and Widgets

Every little thing you include on your web page takes up a request-response cycle, has to be delivered to your page, uses up additional memory, has to be rendered by the browser, and may even require additional compute cycles to process if it includes code.

The classic example, of course, is the Flash animation. Remember the old days where not only would the site interrupt your browsing to show an animated introduction, but you'd also end up waiting while the progress bar made slow transfer and long load times painfully obvious?

Today, over-the-top, performance-killing Flash, ActiveX, and Silverlight presentations are mostly a thing of the past. Occasionally you'll encounter a site that uses a plug-in—most often Flash—but the lack of plug-in support in mobile browsers is quickly making them irrelevant except for very specific use cases such as gaming. Hopefully your site doesn't require visitors to install a plug-in before they can do anything interesting (good luck making that an enjoyable experience for the average user).

Ads and Video Have a Cost

Ads and automatically playing video seem to have taken up the crown of annoyance and spoiled performance, particularly when implemented poorly. Ads are typically images or Flash, often fetched dynamically from a remote server by a script.

Let's count the ways that ads and video can be slow: code has to load and run, kicking off a request to a remote server; the server does some work and sends back some data. If you're lucky, you get a few kilobytes. If you're not, it's hundreds of kilobytes, which gets interpreted by a plug-in running separately in the browser.

If you're really unlucky, a stream of video comes flying over a slow wireless connection, loads up a separate playback plug-in, and starts immediately playing at full volume while you're in a crowded public setting.

Slowed Down By Code Behind the Scenes

It gets worse: if the code behind these ads runs before the page is rendered completely, but hangs up due to a slow response from the ad server, the entire page will remain unresponsive. This continues until the ad eventually loads, the process times out, or the user closes the browser tab and moves on (which can happen in as little as 3 seconds).

Tim Kadlec—a web developer, consultant and author of books on web design practices—started a recent blog post with the question "How fast is fast enough?" and offered some interesting data points from a

1968 study on response times: 100ms feels instantaneous, while 1 second feels uninterrupted. After 10 seconds you've lost the user entirely.

How does your site fare against those metrics?

Weighing the Useful Against the Wasteful

It's easy to turn up our noses at poorly performing plug-ins and advertising networks. Flash is so ten years ago, right? And ads are just something we put up with so that everything can be free.

But the reality is not so simple; plug-ins still power some significant, well-liked businesses that stream video and music. Online advertising is a $100B business. Great web experiences and growing businesses depend on implementing these features the right way.

There are other, less obvious pieces to this puzzle as well. Social media widgets, for example, can clutter and weigh down a page if used indiscriminately. However, as I mentioned previously, social referrals can be a powerful tool for customer engagement.

Software as a Service (SaaS) plug-ins are another example. Some of these, like Disqus, can be incredibly valuable services for community building and support without having to invest in creating and maintaining your own backend infrastructure. However, if integrated poorly into your site, they can slow performance down and even make your page unusable if the service becomes unavailable. How ironic if the feature you choose to attract community actually turns it away.

Cut and Paste Development

When even simple, helpful components of your website start to become performance bottlenecks, you have to ask whether development techniques themselves are the culprit. The Web has come a long way in 20 years, and new tools and frameworks make the process of building ever more sophisticated websites easier and easier.

From the basic components up to the code and database, building a website is practically a point-and-click affair these days. There are tools like Yeoman to build out the scaffolding, an ever-expanding universe of MVC frameworks, jQuery and similar code libraries to simplify and expand JavaScript, and more.

The Easy Route Makes Solving Problems Difficult

Ultimately, the problem boils down to web development techniques that err on the side of easy instead of correct and optimized for basic performance.

Frequently, we choose the solution that gets work done quickly and "efficiently," and runs well enough on our test environment. It's a reality that developers need to get the work done now, not spend a week researching the best solution. Premature optimization is the root of all evil, right?

Less experienced developers fall into this trap as well, choosing cut-and-paste or easily plugged-in solutions over learning how their programming environment really works. If you do a quick search online for a coding solution, you're not likely to find something up to date on the first page of search results. There are a lot more out of date coding examples with established Google cred you'll have to sort through before finding the most current ones. Discerning the good from the bad can be difficult if you lack experience, time, and patience.

Shiny New Things Clutter the Solution

For experienced developers, it can be equally easy to fall prey to the temptation of shiny new things over tried and true (and unexciting) solutions. When you've been building websites for a while, the simple solution can start to seem boring. New libraries and frameworks are coming out all the time. Sometimes new tools make drudge work easier; other times they enable new features. Frequently learning something new and different spices up an otherwise dull assignment.

Some of my friends are hot on Polymer to create polyfills and custom elements and to manipulate the mysterious Shadow DOM. It's neat stuff, but very cutting edge, and browser support is definitely not guaranteed.

Or how about KerningJS, a JavaScript library just for tweaking font kerning in your CSS?

For many years, jQuery was derided as a crutch for the lazy or untrained. That is a largely unjustified accusation—like any tool, it can be abused, but jQuery simplifies what can be time consuming to code with relatively little additional performance cost.

I can't make the claim that these tools are bad ideas. Some of them are ingenious, and there's a correct time and place for putting every one of them to use. However, your site may not be that place. If you have clear goals for your web application—and performance is one of them—that should help you choose which tools and techniques will be most effective and which are distractions.

I'm going to argue that a good first step is to put away the shiny and the new, and instead focus on the tried and the true. Keep it simple.

With that thought, let's take a look at some straightforward answers to the development side of this thorny performance problem.

Simplify, Then Add Lightness

We'll get to some more technical solutions later, but it's worth starting out with some basics. Colin Chapman, the legend behind Lotus, the sports car manufacturer and racing team, summed up his engineering philosophy as "Simplify, then add lightness."

It's just physics. All else being equal, a lighter car accelerates more quickly, goes around corners faster, and stops more efficiently. A simpler car is easier and cheaper to build, more robust, and faster to fix when it does break.

Or, to simplify and lighten my own example, a scout leader would tell us when preparing for a backpacking trek: "If you watch the ounces, the ounces watch the pounds."

If you're starting a new project, begin with a simple, balanced roadmap for development. Consider what's most valuable about your web app and focus on the solution best suited for that aspect of the project. Avoid the kitchen sink approach! Choose a solution and a set of tools and implement them well. That may mean spending additional time evaluating the technical implementation details of your approach and truly understanding and accepting the tradeoffs.

At each step along the way, consider the really basic aspects of your site implementation. Do you need all that JavaScript? Are you packaging and loading your CSS and code dependencies efficiently? Can you make the images smaller or optimize them? Which social widgets get used most often? Can you leave others off? Does anyone actually print your page anymore? Do you need special CSS and JavaScript to create a view just for that audience?

Optimizing Your Optimization

Performance optimization is a technical task, but it may help focus your efforts by putting it—at least initially—in a business context. What are the most popular, most requested pages on your site? These pages are probably the first you should be looking at closely and potentially optimizing.

Consider also the pages you think are most important to your business, whether that's the center of ecommerce transactions, interaction with your customers or audience, or the showcase for your expertise. If there's a delay in rendering or interaction here, you absolutely know you're losing visitors… and probably revenue.

If you can put a dollar amount on traffic or interactions (whether that entails reading your free advice, downloading your software, or making a purchase) on a page, you can start to calculate what delays and poor performance cost. Then, balance that against the long term value of investing in development time to slim down your site and improve performance. Remember that you can correlate delays in site loading and interactivity with increases in rates of user abandonment. Put real numbers on those delays to figure out what they may cost. For every day you wait to speed up your page, you can figure the potential lost revenue.

Looking for Performance In All the Right Places

It doesn't take a huge effort to start tracking down performance bottlenecks. The two most important, which I've touched on already, are rendering time (when is the page fully displayed in the browser?) and time to interaction (when can the user actually do things on the page?). We've all experienced these issues before: how many times have you seen elements bounce around a page as new ads and widgets load up. You can't read that page, much less do anything with it.

Twitter famously called this optimization "time to first tweet" when they did a round of reconstructing the site experience back in 2012. The Twitter engineering team blogged about the changes at the time, as did team member Alex Maccaw. I'll get to some of the interesting tips they shared later. Relevant to the current discussion about finding bottlenecks, however, was the team's use of the W3C Navigation Timing API. This is a W3C specification that lets you create your own scripts to measure latency within your site.

There are a variety of tools available that let you benchmark real-world page speed and interactivity. A few that come well-recommended by professionals in the business include Google PageSpeed Tools and WebPageTest.

A Real-World Example: SpeedCurve

SpeedCurve is an interesting site that breaks down many elements of your page's loading and rendering behavior in ways that highlight where your code is inefficient. (It's basically WebPageTest with a better UI and features for regular testing added.)

There are rendering timelines that show step by step, in visual snapshots you can't ignore, how the page renders over time—often several seconds (see Figure 6-1 for an illustration of this). You can see the number of individual requests made to servers for content, and you can see how response time varies by client platform, including mobile devices.

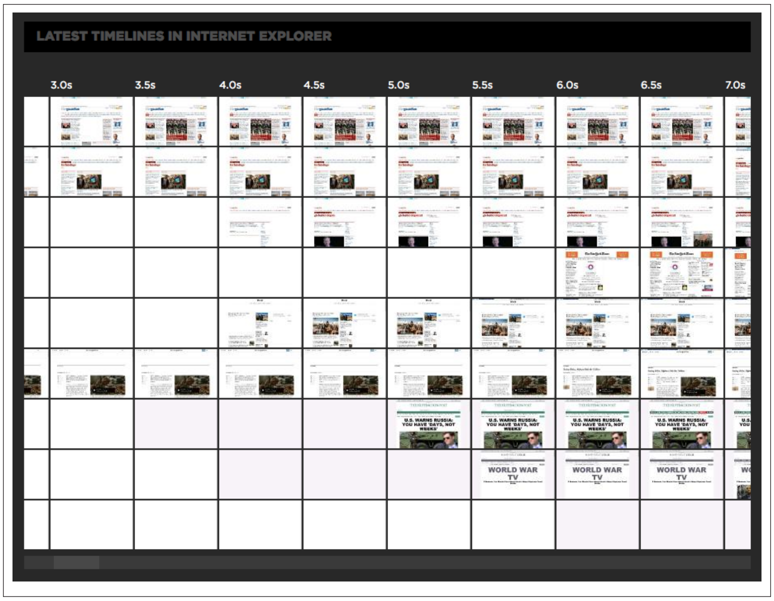


Figure 6-1. SpeedCurve and other tools display rendering status in a timeline

SpeedCurve is a commercial service, and it may be worth the price for convenience and a nice set of features in a single package. If you prefer to do it yourself, however, some of this can be accomplished with tools built right into current browsers. Getting to know the developer tools is a great first step for quick analysis of site performance.

DIY Performance Testing

Want to do your own back-of-the-envelope testing? I'll lay out how to do this using Chrome:

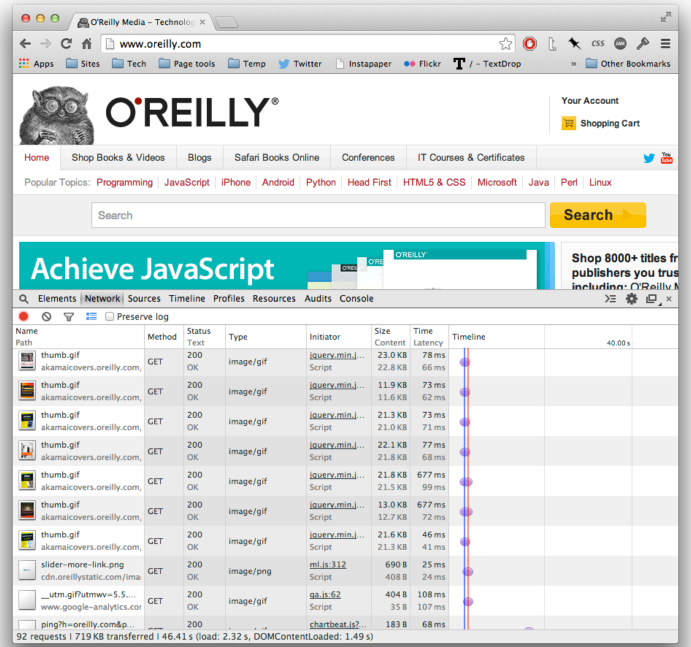
- Go to the site you want to test
- Open the developer tools using Cmd-Option-I or Tools → Developer Tools
- Click on the Network tab
- Reload the site using Cmd-Shift-R on a Mac or Ctrl-F5 in Windows

Holding down Shift on the Mac or Ctrl in Windows makes the browser bypass the cache and request every resource from its specified URL. The Network tab lists every resource loaded, with information about the request. The status bar shows a summary of requests, data transferred, and time elapsed (see Figure 6-2).

Hopefully I don't have to mention that the data is only relevant if you're measuring a deployed site. Measuring performance when loading the site from a dev server or localhost won't tell you nearly as much.

Another approach is user timing and real user monitoring. It's hard to be objective when clicking around your own site, so get someone else to do it. Get out a stopwatch and observe as the person clicks around. Note how long rendering takes and when the user can start tasks (and which ones she chooses). Note when she seems frustrated and when she clicks away.

These are really simple tests to run, and they don't even require adding time and tooling to your development environment. Firefox, Internet Explorer, and Safari all have similar tools, so you can do this testing in your favorite browser.



Name Path	Method	Status Text	Type	Initiator	Size Content	Time Latency	Timeline
thumb.gif akamaicovers.oreilly.com,	GET	200 OK	image/gif	jquery.min.j... Script	23.0 KB 22.8 KB	78 ms 66 ms	
thumb.gif akamaicovers.oreilly.com,	GET	200 OK	image/gif	jquery.min.j... Script	11.9 KB 11.6 KB	73 ms 62 ms	
thumb.gif akamaicovers.oreilly.com,	GET	200 OK	image/gif	jquery.min.j... Script	21.3 KB 21.0 KB	73 ms 71 ms	
thumb.gif akamaicovers.oreilly.com,	GET	200 OK	image/gif	jquery.min.j... Script	22.1 KB 21.8 KB	77 ms 68 ms	
thumb.gif akamaicovers.oreilly.com,	GET	200 OK	image/gif	jquery.min.j... Script	21.8 KB 21.5 KB	677 ms 99 ms	
thumb.gif akamaicovers.oreilly.com,	GET	200 OK	image/gif	jquery.min.j... Script	13.0 KB 12.7 KB	677 ms 72 ms	
thumb.gif akamaicovers.oreilly.com,	GET	200 OK	image/gif	jquery.min.j... Script	21.6 KB 21.3 KB	46 ms 41 ms	
slider-more-link.png cdn.oreillystatic.com/ima	GET	200 OK	image/png	ml.js:312 Script	690 B 408 B	25 ms 24 ms	
__utm.gif?utmwv=5.5.... www.google-analytics.coı	GET	200 OK	image/gif	ga.js:62 Script	404 B 35 B	108 ms 107 ms	
ping?h=oreilly.com&p...		200		chartbeat.js?...	183 B	68 ms	

92 requests I 719 KB transferred I 46.41 s (load: 2.32 s, DOMContentLoaded: 1.49 s)

Figure 6-2. Using Chrome's developer tools to measure website performance

Site Optimization From the Top Down

Once you've determined that you have a performance problem and have narrowed down where you need to focus optimization efforts, it may be tempting to simply throw additional resources at the computation end of the solution. You might be tempted to tackle performance issues with more servers, load balancing, caching, and so on.

In some cases, this is the solution. But before you head too far down this route, consider whether you're just temporarily putting off development work and instead adding additional cost and complexity to keeping your website running. This is technical debt of another kind. You're actually adding ongoing system administration work instead of fixing the root of the problem.

Digging into the code and related page resources—HTML, CSS, JavaScript and images—isn't actually that much work. You're not committing to a full site redesign. Instead, there are many simple checks you should be doing anyway to optimize all the elements of your site. Developing a better understanding of how the browser pulls in and renders all of your page resources can help, too.

Simplify Your HTML

Your first step is optimizing the HTML itself. It's a good idea to go through your basic HTML markup and make sure you're following modern conventions. Start by making sure your code is clean, readable, and not using any unnecessary tags.

Dave Raggett's HTML TIDY was built to automatically check and clean up HTML, and updated, web-based HTML cleaners use it (and a few similar tools). Check out Cleanup Html and Dirty Markup, two examples of the variety of tools available to check CSS and JavaScript as well.

I also recommend getting on board with semantic markup: separating content from its styling and stripping the HTML down to only the required elements will help you get organized. In 2008, Chris Coyier wrote an excellent primer called "12 Principles For Keeping Your Code Clean." Most of the principles still apply.

HTML5 brings new features that enable you to slim down semantic markup even further. There's a new, simpler doctype, and new tags like `<header>`, `<nav>`, `<section>`, and `<footer>` that map directly to elements of your page. This means more straightforward element tagging and potentially fewer `class` and `id` declarations. Josh Lewis has a helpful article called "HTML5 Semantic Page Structure" that updates some of the information from Chris Coyier's piece. I highly recommend reading these introductions to the topic if nothing else, but this is modern web development information you should know.

Put Your CSS and JavaScript on a Diet

Externally referenced CSS and script files cause additional resource requests and roundtrips, but can be cached by the browser. Even small files incur round trip costs that cause latency, whether or not bandwidth is a performance factor. DNS lookup also causes latency, so minimize the number of different DNS lookups needed to load critical resources. Further visits still need to request the data, though it can be fetched from the cache, which is faster.

But remember: you only get one chance at a first impression. Make sure that initial experience is a good one.

Minification makes externally referenced files smaller and more efficient. Combining your CSS and JavaScript into as few files as possible also reduces the number of requests. Remember, each file you need to load requires its own request and response (and potentially a DNS lookup as well). Don't use the CSS `@import` directive as a shortcut to combine stylesheets—it doesn't work like compiling code; the browser still needs to request both files.

A Little Order Keeps Requests from Blocking

The order in which you reference CSS and script files is important. The general rule is to reference styles in the header and JavaScript at the end of the file. This is great advice, but why? Because browsers wait to render content after a `<script>` tag until the script finishes running.

If the script has to be requested and downloaded first, add that time to the equation. This delay is intentional to allow the script to make any potential changes to elements in the page or their layouts.

While the script is downloading and executing, any resources beyond the associated `<script>` tag in the page are effectively blocked from action. That includes downloading images, CSS files, or any other scripts. Note that inline scripts block, too, so be careful where you put them.

A script effectively blocks, if only temporarily, any further action when it is encountered. Now you see why common practice has been to load CSS in the header and JavaScript later, so the majority of the page resources can download and render before the script blocks.

Loading JavaScript Asynchronously

Another option to consider—whenever possible—is loading JavaScript asynchronously. This instructs the browser to download a referenced script file, but wait on executing it until later when the UI thread is not busy rendering the page or handling user input. This is a common technique for loading analytics code, for example, and can also be used effectively for non-layout scripts, loading third-party scripts that could be slow or offline, and even handling below-the-fold DOM manipulation.

One old method you can employ is the `<script>` tag's `defer` attribute:

```
<script defer src="http://yourJavaScript.js"></script>
```

Most browsers you encounter today support the `defer` attribute. Some older versions of Opera and Safari did not, unfortunately, and Internet Explorer prior to version 10 offered only partial support. The catch for all supporting browsers is that your script can't do any DOM manipulation or use the `document.write` method when you employ the `defer` attribute.

If you can target them, more current browsers support simply using the HTML5 async attribute when referencing scripts.

```
<script async src="http://yourJavaScript.js"></script>
```

This has all of the advantages of the defer attribute with none of the restrictions.

If you need to support older or non-conforming browsers and defer won't do the trick, the traditional script DOM element technique may work. It requires more code, but also gives you extra flexibility:

```
<script>
    var resource = document.createElement('script');
    resource.src = "//third-party.com/resource.js";
    var script = document.getElementsByTagName('script')[0];
    script.parentNode.insertBefore(resource, script);
</script>
```

I borrowed this code from Chris Coyier's excellent "Thinking Async" post at CSS-Tricks. I would definitely recommend using this article as a resource for understanding async JavaScript. Coyier provides further explanation of how the script DOM element works along with handy tricks related to asynchronous script execution, calling ads, incorporating jQuery, social media scripts, and more.

More Tips for Optimizing CSS and Script Resources

This is also a good time to refer back to Alex Maccaw's blog post about re-engineering the Twitter site experience, which I mentioned earlier. Maccaw offers some excellent advice related to simple optimization of your CSS and script resources—as well as caching them effectively for return visits—that I highly encourage you to read.

None of the recommendations I've covered here are particularly groundbreaking and should be well known to experienced web developers. However, they're worth revisiting. Simple optimizations are often the first and easiest to forget in the heat of getting things shipped.

Is an Image Worth 1,000 Bytes?

Early on, I started slamming large image files as a problem, so I should probably spare a few words here on a solution for handling them. I discussed size and resolution in the context of high-density Retina displays. You have options regarding which images can best be resized to 1.5× or 2×. Fortunately, you also have a variety of supported image formats to choose from and excellent tools for optimizing image quality (in terms of bytes) regardless of image size (in terms of pixels).

A very simple recommendation is to avoid using images as construction elements of your page. In other words, don't use a 50×100 pixel image to create your header if it's mostly white space. Instead, crop the image to the essentials and use CSS to locate elements on your page. It's not only the right way to do the Web, it's also far more efficient.

Efficient Image Formats Save Space

Choose the image format and color palette that is appropriate; JPG files can be configured to optimize size while maintaining excellent image quality. For the flat design popular today, you can often employ 8-bit GIF or PNG files with optimized color palettes to get tiny files.

Smush.it is a free image analyzer and optimization tool that's worth a look. Just upload your images (or provide URLs if they're already hosted on your site) and the service provides a zipped download of the optimized files, telling you exactly how many bytes it saved.

Both JPG and PNG offer "progressive" or "interlaced" modes that enable you to display a low-resolution version of the image immediately while the full resolution image downloads. The displayed image progressively increases in quality as the bits come in over the wire, hence the name. Generally speaking, this technique is frowned upon. However, there are always tradeoffs worth considering for specific user experiences you may find desirable.

Used carefully, progressive images can help. Remember, perceived speed is important to user experience. If the low-resolution image displays immediately, even though the full image isn't there yet, your site appears to be fast. If your site or business relies on excellent image quality—professional photography is the obvious example—you can take advantage of this feature to get the best of both worlds. Just make sure to use it wisely on only the most important images!

An Old Trick That Still Works: CSS Sprites

CSS sprites involves combining several images into a single file then using some clever CSS to display only specific slices of that image in specific locations on your page. This technique is particularly handy for small, regularly shaped image elements like icons, where you might fit dozens, organized into a grid, in the image. See an example from Brandon Setter's blog in Figure 8-1.

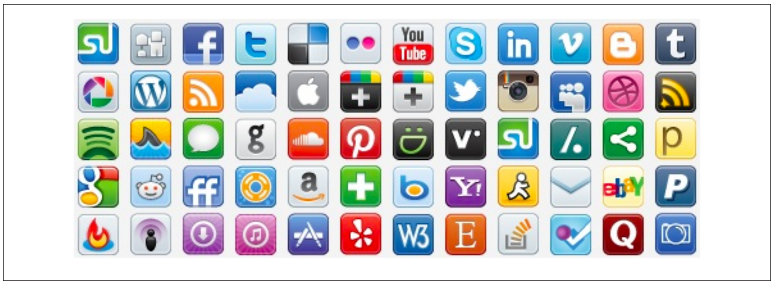


Figure 8-1. An example of CSS Sprites within a single image

The downside is some rather fidgety image and CSS creation, though this technology is old enough that you'll find many helpful utilities to smooth the process. The upside is that you get one single request, response, and download instead of dozens. For a handy tutorial, see Smashing Magazine's article "CSS Sprites Revisited", which contains references to even more ancient discussions of this evergreen topic.

Mobile Doesn't Mean Speedy

The Web experience on smartphones and tablets isn't going to make your life any easier. Web browsing on mobile devices accounted for over 20 percent of traffic at the end of 2013 and as I pointed out earlier, mobile ecommerce is growing at a rate of 14 to 17 percent annually.

If you think about it, mobile devices with great web experiences are the ultimate shopping weapon: as soon as shoppers think about needing an item, they can whip out a phone, browse to the seller's site, do a little research and make a purchase… assuming the site performs well enough to close the deal.

The Tammy Everts blog post I referred to at the beginning contains some important research points about mobile performance. Over 80 percent of mobile shoppers expect site performance equal to or better than desktop browsing, and shopping cart abandonment rates rise to 97 percent, versus 70-75 percent on the desktop.

Mobile is Always a Slower Experience

A significant problem is performance expectation: consumers think mobile web browsing will be faster. Technically speaking, that's almost impossible to achieve today.

Under optimal conditions, mobile devices using WiFi or LTE networks can achieve latency times almost comparable to desktop broadband, at least for general browsing. However, not all phones and networks are created equal. FierceWireless and OpenSignal tested mobile network latency across carriers and networks and found that, while some LTE networks provided low mean latency, other offerings could be considerably slower.

Be Careful How You Optimize for Mobile

At one time it was considered smart to create a separate, slimmed down mobile site (often called an “m.dot” site) to accommodate mobile users. Aside from the hassle of creating and maintaining two semi-parallel design and development tracks, there’s an inherent performance killer baked into the concept: resolving a connection to the main site, detecting the mobile browser, redirecting to the mobile URL, and starting to transfer those files adds several rounds of request-response-download latency to an already slow connection.

Responsive web design that seamlessly falls back to the best experience for a given screen size and connection is a far better solution. Add to that some of the basic site optimizations I’ve already discussed and you’re most of the way to having a site that’s inherently built to perform well whether on a desktop browser or a handheld device moving at warp commute speed.

Another aspect of optimization for your site that can grow out of consideration of mobile users is—going back to business-optimizing goal setting again—thinking at the design level about what’s most important to your site. Fast and simple seems to be the best experience in mobile.

For a deeper look at this subject, take a look at the Chapter 8 preview of Ilya Grigorik’s book for O’Reilly, High Performance Browser Networking.

Next Steps

Now you've seen that there are clear industry trends toward faster, better performing websites contributing to better, faster-growing businesses. And yet websites among the most prominent online retailers continue to grow bigger and slower every year.

I've also shown that you don't have to follow this trend. In fact, steering your site in the opposite direction—toward a smaller, faster design and development profile—can pay dividends for your business. Looking back through the ideas I've presented, you can lay out a straightforward checklist to follow before diving into more complicated, advanced optimizations:

1. Start by measuring the size, request latency, and load time of your current site on various devices. You can do this right in the browser.
2. Use online tools to measure more advanced site performance metrics and compare them against industry averages.
3. Know how long your page takes to finish rendering and what the time is to the first interaction. This is your first-impression metric.
4. Evaluate your business goals and try to correlate site performance with traffic, conversions, revenues, or whatever metric is relevant to your business.
5. Focus optimization toward the most revenue-related aspects of your site.
6. Count all of the components that have to be requested to render your page. Can you reduce this number?
7. Slim down your HTML. Combine and minify your CSS files.

8. Take a close look at where you reference scripts and when they execute. Are scripts blocking page rendering or interaction?
9. Make sure you're using images wisely and optimizing them for size and quality. A little bit of code can make sure you're not sending too many image bytes where they're not needed.
10. Don't throw out social features just because they add to your requests, scripts, and page size if they benefit the business; just be sure to use them tactically instead of spreading them all over the place.
11. Starting with a focus on the mobile experience—most likely your fastest growing customer experience—may help you target performance optimizations that benefit all visitors to your site, even those using the desktop.

To summarize, keep it simple and keep the focus on a great experience for your visitors. A flashy site might seem impressive, but the numbers don't lie: customers want a quick, responsive experience. The sooner you deliver it, the happier they'll be.

About the Author

Terrence Dorsey is a writer, editor, and content strategist specializing in technology and software development. He is currently a Senior Technical Writer at ESPN, working with the Data & Platforms Architecture team on next-generation sports data APIs. He also writes a monthly column for Visual Studio Magazine. Previously, Terrence was the Director of Content Development at The Code Project and a senior editor and columnist for MSDN Magazine and TechNet Magazine. Read his blog at terrencedorsey.com or tweet him @tpdorsey.