

Typical conversations about Web performance tend to revolve around client/server latency, database performance, and the time it takes a server to render and send HTML to the client.

Those processes, however, generally account for only a fraction of the time a user spends waiting for content to load in a browser window. Greater time is spent downloading, caching, and displaying JavaScript, CSS, and images. Presentation layer performance tuning for Web applications really boils down to two simple concepts:

- Request minimal data
- Request fewer times

The goal of this document is to show, by example, how client side Web performance can be increased by extending “fewer requests, less data” to the following four facets, each addressing various aspects of client side web development in terms of how to prepare JavaScript, CSS, and images for deployment:

1. **Compressing** (minimize data)
2. **Caching** (minimize requests)
3. **Merging** (minimize data & requests)
4. **(CDN) Hosting** (minimize requests)

Problem:

With technological innovations such as client-side frameworks, Rich Internet Applications, and AJAX, browser-based processing has become more prevalent, which means slow page load time can affect the user experience.

Solution:

Compressing files, managing cache configurations, reducing HTTP requests, and using Content Delivery Networks can significantly improve the performance of page rendering on the client side.

Benefit:

On-demand, quick responding Web applications can mean the difference between keeping and losing partners, customers, and clients in the highly competitive Web services industry.

Compressing

For large scale, high traffic Web operations – where performance is critical and every unnecessary KB is squeezed out of the equation – compression techniques can have a big impact on performance. For JavaScript and CSS, file compression tools remove comments, collapse white-space and line breaks, and replace symbols with shorter name representations, reducing the amount of data sent by as much as 50 percent. Compressing images tends to be less useful since the final impact is negligible. Because developers prefer to have readable code during development, compression is obviously implemented as part of the final production build process.

Asset compression can be achieved by using a server plug-in for gzip or deflate compression. While sending fewer bytes across the wire is a benefit, the extra CPU time spent processing the data on both the server and client end may offset the gain. Certain settings can be tweaked to achieve the best results, but the inconsistencies of browsers become problematic and some architects are arguing against this practice – calling it an unnecessary resource hog (Cal Henderson/Flickr).

Caching

Caching aggressively is increasingly important for modern Web sites that incorporate extensive use of JavaScript and CSS. Unlike images, which tend to remain unchanged as assets, CSS and JavaScript files are often revised and updated on a regular basis, which creates a problem for owners with cache sensitive sites. The techniques for configuring a Web server are outside the scope of this paper, but the key is to tell the server to cache every URL forever. To let the server know a file has changed, use a different file name. Large Web sites commonly create a change process where a new version number is inserted into a file name (for example, common.v1.css becomes common.v2.css). The links to these files need to be updated programmatically as well, with page language or templating code.

Caching can be further enhanced by configuring the response headers for requested assets. Instead of downloading an asset that has not recently changed, a host server sends a header-expires entity that directs the user agent to use a cached version of an asset, which is set to expire in the distance future. Images usually do not get updated very often, so why request them over and over? Specifying both “cache-control” and “expires” should handle HTTP 1.0 and 1.1 protocols. A typical response header, shortened to show only caching information, might look like this:

```
Date: Tue, 17 Apr 2007 18:39:57 GMT
Cache-Control: max-age=315360000
Expires: Fri, 14 Apr 2017 18:39:57 GMT
Last-Modified: Mon, 16 Apr 2007 23:39:48 GMT
```

A Cacheability Engine can test the data on a particular site. The engine gives you a detailed analysis of the different objects and how they are being cached. Based on the results, you can tweak parameters for the various objects to attain optimal performance.

Merging

The browser's default limitation of two concurrent HTTP requests from the same domain at any time is a rendering bottleneck. Reducing the number of requests can decrease the time needed to render a page. Concatenating client side HTTP resources can also address this particular issue.

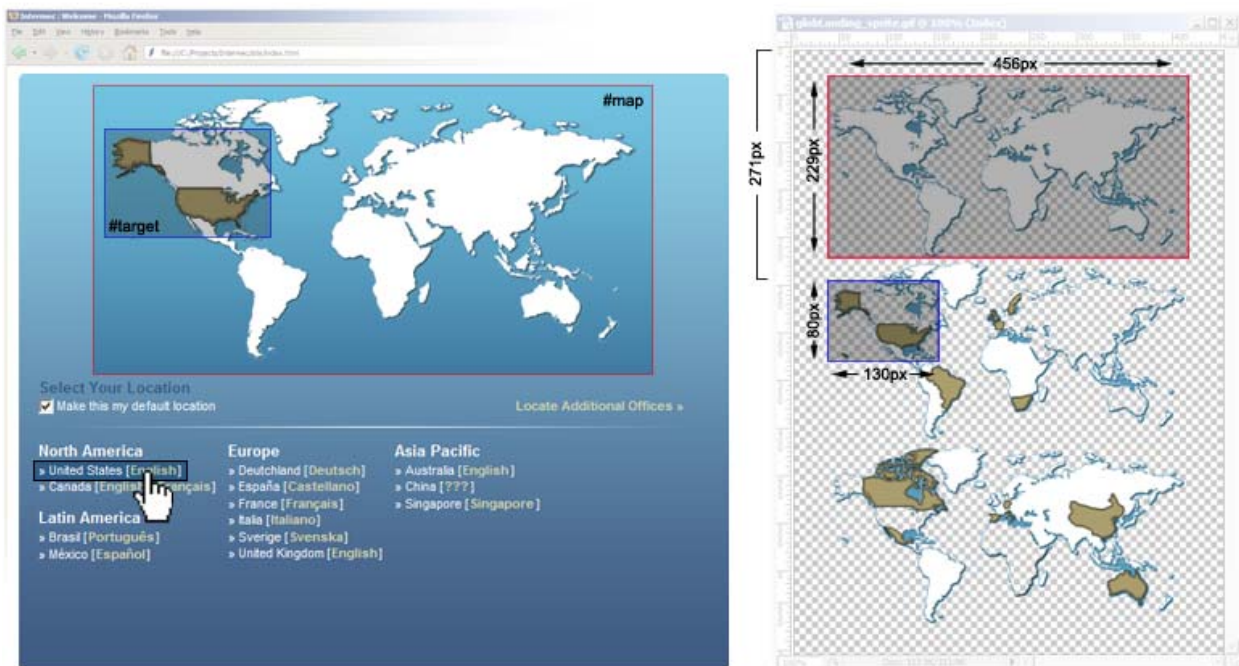
In terms of JavaScript and CSS resources, reducing HTTP requests as a general rule makes for increased performance, although it should be weighed against the potential drawback of having large monolithic files. If these resources are going to change often, which they almost always do for evolving Web applications, there is no benefit to users in having to download 100K plus files every few days. If JavaScript is changed often, parcel out those areas of code that get updated frequently into separately downloaded files. Furthermore, if an application has distinct parts, such that a major portion of JavaScript is used only in some particular area, keep that code bundled separately, layering the application logic into smaller chunks. Since the HTTP request limit is tied closely to the default limitation of two requests per domain, highly trafficked Web sites may employ multiple domains to host JavaScript and CSS, typically on content-delivery-networks (CDNs), which use varying domains.

A popular technique for reducing HTTP requests for large image sets is **CSS sprites**, a relatively new concept that allows style sheets to manage the partial visibility of a single large image that also contains other related images. Sprites were originally used in the early years of the computer gaming industry, when bandwidth conscious game designers laid out all the game graphics in "sprites," one large image that contains many smaller images. The game engine then displayed one part of the image while masking out the rest, requiring only one image to be loaded into memory.



The same can be applied in CSS, where selector attributes can be used to show and hide parts of an image, as needed. This technique is effective in reducing HTTP requests for such common interface functions as image-based button roll-overs and other components that require several images to preload and be ready on an event trigger.

For example, many multi-national companies have Web sites with landing page maps. These interaction models usually involve roll-over states for various countries – a task that is usually suited for image maps or Flash. With the advent of CSS sprites, this type of functionality can once again be built using DHTML, thus providing a more efficient, semantic, accessible, and search friendly user experience. Consider the recent redesign for Intermec – the world leader in manufacturing RFID based technologies. The map itself takes up a significant portion of page real estate. As the user rolls over the text links for language options, the corresponding country lights up on the map. With twelve countries that need a highlighted state for the same map, this poses a problem. Fortunately with CSS sprites, a single HTTP request with an image that contains roll-over states for all the countries involved can be made.



In order to highlight the United States, for example, the user initiates a *mouseover* event on the appropriate object (in this case, a link tag) that triggers JavaScript to animate the opacity of the background image sprite from 0 to 100 for the target.

Essentially, CSS displays a region of an image and masks the rest, using a combination of attribute positioning and hidden overflow. In the case of the Intermec map, the 456px by 705px sprite GIF image has three components:

- The top map is the default view on page load.
- The second and third maps have highlight states for various countries.

The key is to spread the countries around so that no two countries are sharing the same rectangular coordinate space. Intermec used only two additional repetitions of the map. The target's CSS selector has three sets of attributes that define the masking region:

- **Left** and **top** sets the position of the viewing area to the correct country on the default top map.
- **Height** and **width** sets the viewing area's height and width.
- **Background-position** slides the image to the correct position, showing the roll-over state in the viewing area created by the first two masking region attributes.

```
#map {  
  width:456px;  
  height:229px;  
  background:url(images/sprite.gif) no-repeat;  
  overflow:hidden;  
}  
  
#target {  
  position:absolute;  
  left:0;  
  top:35px;  
  height:80px;  
  width:130px;  
  -moz-opacity:0; opacity:0; filter:alpha(opacity=0);  
  background:url(images/sprite.gif) no-repeat;  
  background-position:0 -271px;  
  overflow:hidden;  
}
```

In other words, two HTML elements have the same image sprite set as a background, the first showing the entire map of the world, while the second is masked to show only the roll-over state for the country in consideration. In this fashion, roll-over states for any number of countries can be created with a single HTTP image request by stacking masked versions of the same image – one on top of another.

(CDN) Hosting

CDNs (Content Delivery Networks) have recently been used to reduce the processing and bandwidth requirements of hosted Web applications while increasing performance by

distributing resources across server networks. Proximity to these Web resources directly translates into increased performance.

Some companies, such as Akamai, have built their businesses around providing these networks for many large scale Web operations to serve their content. Akamai's pricing structure is based on the total KB weight of Web pages served, times the number of user requests. Improvements in Web performance can affect the bottom line in a very direct way. For example, one of AA|RF's high profile clients pays approximately \$8000 per month for a 320KB homepage. If a presentation layer developer were to implement performance enhancing techniques that reduce total page weight by 25 percent, the client stands to save \$2000 per month. Remember we are talking about just the homepage in this example. The development time quickly pays for itself!

Yahoo! recently started serving their YUI (Yahoo! User Interface) libraries from their own world-wide server farm – for free! The new service provides:

1. gzip compression (reduced file sizes range from 60% to 90%);
2. Quality cache control, with distance future header expiration.
3. Geography-based file serving from edge computing systems.

In the past CDN services were thought of as just hosting for images and large data files, but today they serve JavaScript and CSS as well. A combination of caching/versioning and distribution of files on a CDN will lead to increased performance results.

Conclusion

Just as HTTP requests can affect the time it takes to render a Web page, so can the combined weight of assets affect performance. Employing efficient, semantic, and architecturally well-designed, client-side programming techniques can help keep file sizes down to a minimum.

For example, JavaScript libraries like YUI, Prototype, and Dojo can be used in building common Web components using streamlined, efficiently written code. Adopting Web Standards based design methodologies can reduce the amount of HTML, CSS, and JavaScript used to render user interfaces (for example, removing the need to use such outdated techniques as the



AA|RF built the Yahoo! Talent Show Web site using all of the performance enhancing techniques mentioned in this paper.

infamous 1x1 clearpixel GIF). Using JSON (JavaScript Object Notation) as returned AJAX data can also speed up process response time by up to 50 percent, since the data itself is returned as a JavaScript array, thus removing the need to parse XML.

Although many of the enhancements techniques described in this paper seem to save only a few kilobytes of page weight, the overall net improvement can make a big difference. The importance of adopting these techniques is especially relevant for high traffic Web sites that receive bandwidth intensive usage statistics.

About the Authors



As a senior presentation layer developer, graphical alchemist, and Web Standards advocate for Avenue A | Razorfish, **Fred Welterlin's** experience and areas of focus include designing, architecting, and programming client-side templates, providing project management and technical recommendations, and developing multimedia-based widgets. Fred has ten years experience working as a user interface designer and developer – responsible for conceptual and practical design development for a wide range of business and industry needs.

You may send your suggestions and questions to Fred at

fred.welterlin@avenuea-razorfish.com

Dan Ruspini is leading the design, development and architecture of front-end technology for AOL's Web Communication Suite. As System Architect, Dan is responsible for delivering the next generation of Web based communication for millions of AOL and AIM users every day. Dan has eleven years experience working as a user interface designer and developer – from start ups to top trafficked Web sites. You may send your suggestions and questions to Dan at dan@ruspini.com

References

- YUI: Performance Research, Part 1 & 2

<http://yuiblog.com/blog/2006/11/28/performance-research-part-1>

<http://yuiblog.com/blog/2007/01/04/performance-research-part-2>

- YUI: Free Hosting of YUI Files from Yahoo!

<http://yuiblog.com/blog/?s=yahoo%21+hosts>

- IEBlog : IE + JavaScript Performance Recommendations - Part 1, 2, and 3

<http://blogs.msdn.com/ie/archive/2006/08/28/728654.aspx>

<http://blogs.msdn.com/ie/archive/2006/11/16/ie-javascript-performance-recommendations-part-2-javascript-code-inefficiencies.aspx>

<http://blogs.msdn.com/ie/archive/2007/01/04/ie-jscript-performance-recommendations-part-3-javascript-code-inefficiencies.aspx>

- Vitamin Features » Serving JavaScript Fast

<http://www.thinkvitamin.com/features/webapps/serving-javascript-fast>

- Efficient JavaScript - Opera Developer Community

<http://dev.opera.com/articles/view/efficient-javascript/>

- On-Demand JavaScript

http://ajaxpatterns.org/On-Demand_Javascript

- W3C Header Field Definitions

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.21>

About Avenue A | Razorfish

Avenue A | Razorfish is one of the largest interactive marketing and technology services agencies in the world. The company helps industry leaders such as Kraft, Dell, The New York Times and Starwood Hotels use digital channels to acquire and service customers. Avenue A | Razorfish's full suite of digital offerings includes online advertising, Web site design and development, email and search engine marketing, emerging media strategies, and enterprise portal development. Its award-winning client teams have a great understanding of customer needs and provide solutions through distinct business disciplines, which include: analytics, strategy, technology, media, creative design and user experience. An operating unit of Seattle-based aQuantive, Inc. (NASDAQ: AQNT), Avenue A | Razorfish has offices in markets across the United States, and global operations in Australia, China, France, Germany, Japan and the United Kingdom. Please visit www.avenuea-razorfish.com for more information.

Avenue A | Razorfish
821 2nd Avenue, Suite 1800
Seattle, WA 98104
Phone: 206.816.8800 Fax: 206.816.8808
For more information please visit: avenuea-razorfish.com