Improving the User Experience in a Fast-Forward World

By Vijay Pallithekethil, Software Tester
March 3, 2015

Recently, after experiencing a "service not available" error while checking my email, my first instinct was to go scouring the web, including sites such as downrightnow.com, to check to see if others were experiencing similar issues and if there was an active outage. Even though the outage was no more than 15-20 minutes, I found myself feeling frustrated and wanting to immediately switch to a more reliable email provider. I started thinking that my life depended on getting access to my mail. As someone who has been on the receiving end of these complaints from users because of slowness, or outage issues, I found it strange how I and my expectations have changed over the years. In a world running on fast-forward, with so many things competing for our attention, delays while doing anything shows up as frustrated tweets, blogs and Facebook posts. In this article, I focus on the need for "The Best User Experience" and some overlooked, but easy, fixes for improving the user experience.

> "ad revenues decrease significantly with increased page load times"

Putting numbers on user frustration, a paper by Amazon in 2007 showed that for every 100 millisecond (ms) increase in page load time for Amazon.com, there was a 1% decrease of sales [3]. Google and Yahoo have done similar studies which show similar results whereby ad revenues decrease significantly with increased page load times [4]. A study by Akamai in 2009 [5], showed that 47% of users expected a page to load in under 2 seconds and, if it took longer than 3 seconds, 57% would abandon the site. In addition to that, one criterion that Google uses for ranking websites is the page load time [6]. More recently, the website for the "Patient Protection and the Affordable Care Act" (a.k.a. ObamaCare) was all over the news due to its inability to handle more than a few hundred users at a time during the initial enrollment period. Since the deadline had to be met to ensure the law's success by enrolling as many users as possible, users were encouraged to visit the website during off hours and be set up in an email queue where users would be alerted when the site had fewer visitors so that they could come back and complete the enrollment process. Also, alternative avenues to enroll users were set up such as allowing users to send in paper applications, setting up call centers and so on. For a federal government with deep pockets, all the extra spending is borne by the taxpayer, but for companies for whom online earnings is a major revenue channel, decreased traffic can be a major embarrassment and a big hit on revenue.

Performance improvements can be broadly classified into two main areas, performance improvements on the backend and performance improvements on the front end. I come from a

background where most of my projects were related to performance improvements on the backend, whether it be capturing SQL processing times, profiling code and capturing method times for each backend functional component or studying disk access and comparing it with observed times by the user on the front end. Front end user time was considered negligible, and not a cause for concern for a performance specialist. Front end engineering was then unknown, while today, front end development teams are ubiquitous at almost all companies with significant online revenue streams as they have become extremely critical in building out the user experience. They serve as the "first impression" for a potential client.

From my experience, many times, especially after backend response times showed to be reasonable, the user experience seemed slightly worse as pages took time to load and users were kept waiting. I used to go around asking my development and infrastructure teams why the user experience seemed okay at best and if there was a way to quantify where the time was being spent. Nobody was able to provide an adequate answer until I came across an article from Steve Souders, Google's Lead Performance Engineer, on the impact of front end processing on the response time [7]. What his research has been able to show is, remarkably, that 80-90% of response time is spent on front end performance and that a significant amount of time was spent on downloading HTML, parsing the page, downloading components on the page, parsing CSS, JavaScript (JS) and, finally, page rendering. Google has also developed Google Page Speed [2] for quantifying the time. This turned out to be an eye opener as I checked out sites that I'd worked with previously and all showed time delays on the user interface (UI) side.

Webopedia [1] defines UI as the junction between a user and a computer program. For a web browser (desktop and mobile), the UI includes the web page (HTML, JS and CSS components) which the user interacts with through a pointer device (touch or click). For native (built-in) apps, it's the touch interface with the buttons being the primary method of interaction for the user. Native applications are applications designed for a particular Operating System (OS).

"we try to do everything through browser plugins"

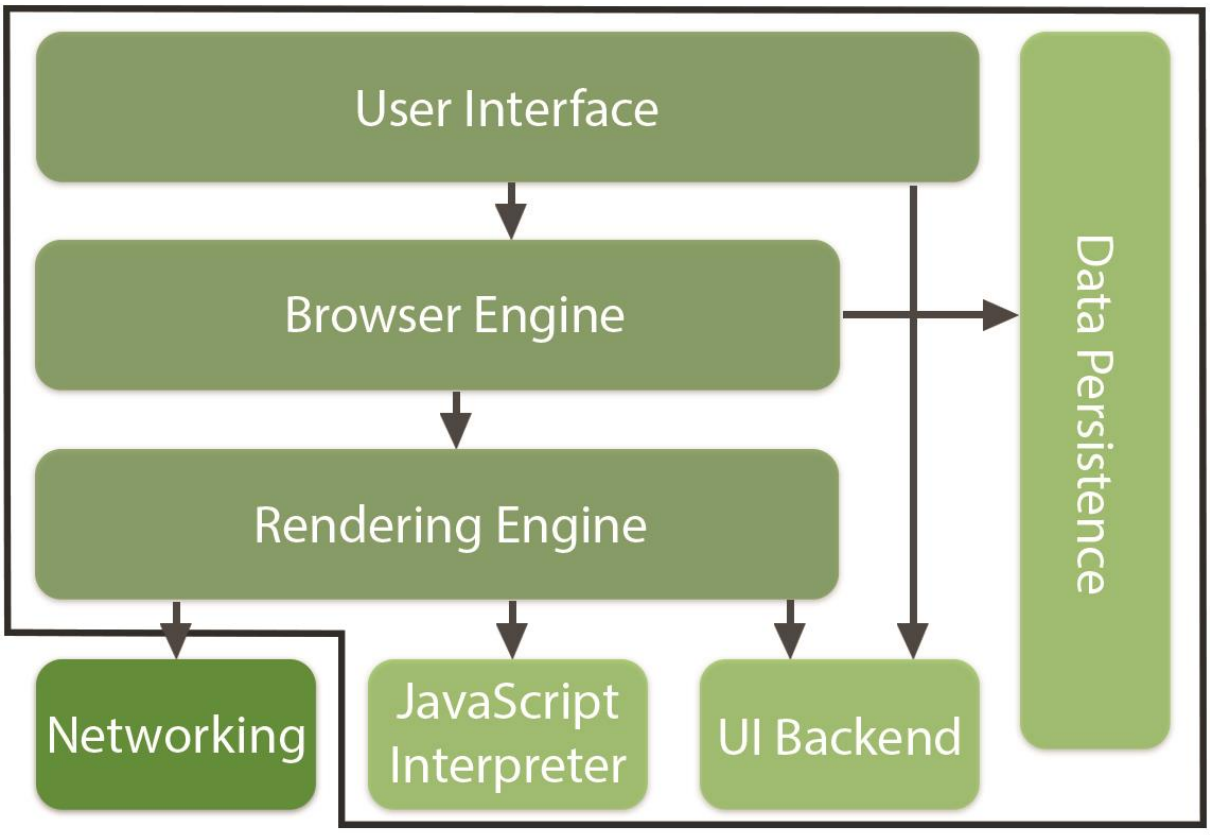
Modern browsers act like mini OSs and, when I use the term mini OS, I am not claiming that it will replace the OS on your computer or all its functions, but trying to highlight complexity. We are trying to do more and more within a browser, resulting in using the browser as a central point for everything whether it be bookmarks, default username/password, plugins for applications such as MP3 players, video downloaders, etc. Instead of installing separate applications for everything, we try to do everything through browser plugins.

At a high level, a browser consists of seven components [8]:

1. User Interface: Includes the address bar, menu options, etc.
2. Browser Engine: Serves as an interface to query the rendering engine and control it

3. Rendering Engine: Responsible for displaying the content (painting the page)
4. Networking: Used for making requests across the network, such as HTTP requests
5. UI Backend: Contains the interface to draw widgets, such as combo boxes, buttons, windows, etc.
6. JavaScript (JS) Interpreter: Parses and executes JavaScript
7. Data Storage: Serves as a persistence layer storing data, a.k.a. browsers own light database where values such as cookies etc. are stored (the actual data repository is the hard disk)

The following is a diagram adapted from html5rocks.com [8], which shows the described components of a browser:



As can be seen from the diagram, even though a browser has seven components, most of the attention is typically focused on performance improvements for the networking part by making sure that all request and response times on the server side meet specified Service Level Agreements (SLAs). While extremely important, there are six other components that impact the user experience. Facebook [9] found that almost half their page response times are spent on JS

interpreting and page rendering, not to mention downloading of all components. Different browsers have different rendering and JS engines which impact performance. More recently, browsers have focused on improving JS engine performance by focusing on features such as multi-stage Just In Time compiling (JIT) and Ahead Of Time (AOT) compiling.

> "once a user sees the initial page load as slow, they don't seem to want to come back"

When a user sends a request through the UI, an HTTP request is sent through the networking layer, and the networking layer responds with an HTML document. The rendering engine receives the reply from the networking layer, and starts parsing the HTML document, converting the different elements to Document Object Model (DOM) Nodes in a tree. The engine parses the style data, both in external CSS files and the style elements. Using this information, a render tree is constructed. After the render tree is constructed, it goes through a layout process where exact nodes are given exact coordinates where they should appear on the screen and then, the final step is the painting of the page, where, in this stage, the render tree will be traversed and each node is painted using the UI backend layer. The rendering engine is single threaded and only the network processes are multi-threaded. Network operations are performed by several parallel threads and the number of parallel connections can be set; different browsers (user agents) have different allowances for parallel downloads (ranging from two to eight). For a chart on the maximum parallel downloads per browser, you can find it in [25]. To get an excellent review of how browsers work in detail, you can find it in [8].

Most web developers and infrastructure engineers rest on the assumption that the slowness observed on the front end would be negated with client side caching, as most users would have all their components downloaded during the first visit to their site. The problem is one of perception, because once a user sees the initial page load as slow, they don't seem to want to come back. As mentioned previously, for any company where their online presence serves as their main revenue channel, this is a huge performance and revenue hit.

Now that we have established the need for performance improvements on the front end, following are a few thoughts to try to improve front end response times:

1. Leverage browser caching by using expire headers [10, 11]: Downloading resources every single time that a webpage is visited is slow and expensive in terms of time and bandwidth. If these resources will be used again, it would be a waste to download the resource content once more if it hasn't changed much.

   While this would seem obvious, a lot of static content is not set up with expire headers. Expire headers will help the browser to determine if a resource file has to be downloaded anew from the server or obtained from the browser cache. While this will only help if the user has already visited the website at least once, subsequent page

loads will be faster. As of July 2014, most modern websites have an average of 90 downloads per page with a page size of 1829 KB (1.8 MB). As the size and number keep increasing, even if the resources are downloaded within fractions of a second, the numbers add up, and it is absolutely essential to reduce the number of HTTP requests for each page.

Expire headers are set for files that don't change often, namely images, JavaScript and style sheets. Of these, only the style sheet files are changed most often and JavaScript and images are changed less frequently. Expire headers can be set in the htaccess (HyperText Access) file found at the root of the website.

2. Using a Content Delivery Network (CDN) [12, 13, 14]: CDNs allow for wider distribution of static content, allowing static content to be pushed to networks closer to the user. This will allow for faster downloading of content for the user, thereby reducing page response times. In addition to that, CDNs improve global availability and reduce bandwidth as you don't have to host the content, reducing the need for you to store static content to deliver to the end user's browser. If the user is a mobile user, there is even more uncertainty as a lot depends on the speed of the wireless connection.

   Having said that, if all your users are local to your area where your site is hosted, then CDNs may not be of much use to you. Cost is always a prohibitive factor but if your site has users around the world, using a CDN is very beneficial in reducing page times. Examples of popular CDNs are Akamai, Limelight networks, Amazon Cloudfront, EdgeCast [26], etc. CloudFare and BootstrapCDN (to name a few) are free CDN providers [26].

3. Avoiding redirects or minimizing redirects [18]: Redirects on a page lead to additional HTTP requests/responses and delays for loading the web page and, therefore, roundtrip latency. This happens when the page has been moved to a different location, if a different protocol is used (for example, changing HTTP to HTTPS) or if you want to direct a user based on their geolocation, language or device type. It is essential to minimize the use of redirects so that delays are minimal.

   Have the application update URL references as soon as the location for resources are changed so as to avoid a costly redirect. Ensuring that redirects happen on the server side instead of having the client redirect will avoid the extra HTTP call. Also, avoid multiple redirects from different domains. For example, if you wish to search something on Google, the only options that show up are based on region/country for the most part (google.ca, google.cn, google.fr, etc.). They don't use googlesearch.com even though their main business is search. Google has become synonymous with search, and it's become so common to mean search, that both Oxford and Merriam-Webster added Google to its dictionary in 2006 [32]. The reason companies normally allow redirects from multiple namespaces is to save namespace and prevent others from taking it and, also, for giving users flexibility. But it

leads to more costs in the end in terms of buying and maintaining the additional namespace(s) and can lead to the user confusing your business name with your actual business.

4. Compress your website [19]: Compress all static content on the website using Gzip, as the browser will uncompress the content by itself. Gzip reduces the size of your website files just as you compress files on your hard drive using a zip program. When a user visits your website, the server responds with gzipped content and, as soon as the browser receives it, the browser will automatically unzip the files to display them to the user. Gzip can be set in the htaccess file found at the root of the website. Because different web servers have different settings for this, I would recommend Patrick Sexton's article [27] to find out the setting for the web server that you are using in order to properly set up Gzip.

5. Reduce the number of CSS and JS files and minimifying code [18, 21]: Try to combine your JS and CSS files into fewer files or else the number of HTTP requests/responses for each file will add to the overhead and, in effect, response times. Even though the payload for your file is a lot larger than splitting it up, it will all be downloaded in one go.

   Since size matters, remove unnecessary characters from your CSS and JS files as this can be done without changing any functionality. The characters that can be removed are white spaces, new line characters and comments. These are for readability but are not needed for execution. This process is called minimifcation of code [28].

   Tools such as JSMin [29], JavaScript Minifier [30] for minimifying JavaScript and CSS Minimifer [31] can be used for CSS files.

6. Specify image size and character response type in your HTTP header [15, 22]: If you don't specify the size of the image when the page is being painted, the browser will first paint the HTML and then will resize for each image based on when the image is downloaded so it will keep painting the page as the images are downloaded one by one to make it fit.

   Also, in that same vein, specifying the character set/MIME type that your website uses in the HTTP response will avoid the browser having to figure it out as it parses the content of your HTML because, otherwise, the browser is left with the task of comparing character types to figure this out. This can be done by making sure that your server adds Content-Type header field to all headers.

7. Move CSS to the top and JS to the bottom [23]: From our discussion on how browsers work, we know that by using CSS, the render tree is built and then the page is painted. Unless CSS is parsed, the page won't be painted. In order to give a faster response to the user so that the layout of the page happens as soon as possible, it is very helpful to move CSS to the head or to the top of the page.

As soon as the browser sees the <script> tag, it stops loading the rest of the document and will wait until the script is loaded, parsed and executed. Since JS scripts are blocking and are loaded synchronously on most browsers, if possible, move them to the bottom of the page. This way, other components of the page are downloaded and JS will be parsed last. Also, use the DEFER attribute when possible with the script. It is now supported on almost all browsers as it has become a part of the specification since HTML 4.01. If other scripts are dependent on a particular JS file, it will be executed only after the entire page is parsed.

8. Persistent Connections [16]: If a lot of connections have to be opened and closed for downloading each and every file on a webpage starting from the HTML, it will add to the response time as web pages contain a lot of files. So in order to save time doing a TCP handshake on each request, it is extremely useful to be able to use one connection for the entire conversation. For this purpose, HTTP Keep Alive needs to be enabled both on the browser and the server to allow for persistent connections. All browsers, by default, use persistent connections these days. However, on the server side, this is not necessarily enabled at all times. So the web server may close connections as soon as the first request is complete. You can instead set up Keep Alive to the server htaccess file for persistent connections.

9. CSS Sprites [17]: CSS sprites have been around for several years now but the concept is well worth repeating because of the tremendous benefits it brings for performance. A CSS sprite is essentially one large image file which contains all the images for your page. So when a user goes to your web page, when an image is to be displayed on the screen, only the one image file is downloaded. This saves both bandwidth and time as there is just one HTTP request response instead of having multiple HTTP requests/responses for each image.

   All images are hidden by default and, to display a particular image on the screen, all that is to be done is list the image from the CSS sprite file and the position where it is to be displayed.

10. Predictive browsing/Pre-browsing [24]: Steve Souders from Google has done a ton of research on the use of standard link prefetching, dns-prefetching and prerendering for various links with the assumption that these will be the next target for the user. The drawback is that this feature will not work on all browser versions and if the user chooses not to go to the prefetched link, it's a waste of resources.

    In prefetching, the browser assumes that a user will go to a certain page, fetches that page and, while in the case of a dns-prefetch, Domain Name Service (DNS) information for the page is collected and stored. In the case of pre-render, the browser renders the page and stores this information within the browser cache so that as soon as the user selects the link, the user is immediately presented with the page.

11. Using Ajax: Web 2.0 should be leveraged as much as possible. The use of Ajax reduces page response times because simple content changes can be made to the already loaded page with a JSON request/response versus loading the entire HTML once again.

This article only covers some of the easier ways to improve the user experience with minimal changes to your site. Yahoo and Google have done tremendous amounts of research in this area and I would highly recommend all the ways they have suggested for improved performance for your website by checking out [33] and [34].

[1] http://www.webopedia.com/TERM/U/user_interface.html
[2] https://developers.google.com/speed/pagespeed/insights/
[3] http://ai.stanford.edu/~ronnyk/2009controlledExperimentsOnTheWebSurvey.pdf & http://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-29.ppt
[4] http://glinden.blogspot.ca/2006/11/marissa-mayer-at-web-20.html & http://www.slideshare.net/stoyan/dont-make-me-wait-or-building-highperformance-web-applications
[5] http://www.akamai.com/html/about/press/releases/2009/press_091409.html
[6] http://googlewebmastercentral.blogspot.ca/2010/04/using-site-speed-in-web-search-ranking.html and http://searchengineland.com/google-now-counts-site-speed-as-ranking-factor-39708
[7] http://www.stevesouders.com/blog/2012/02/10/the-performance-golden-rule/
[8] http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/
[9] https://www.facebook.com/notes/facebook-engineering/making-facebook-2x-faster/307069903919
[10] http://www.websiteoptimization.com/speed/tweak/average-web-page/
[11] http://gtmetrix.com/add-expires-headers.html
[12] http://en.wikipedia.org/wiki/Content_delivery_network
[13] http://www.webperformancetoday.com/2013/02/22/aaron-peters-turbobytes-why-all-cdns-are-not-created-equal-podcast/
[14] http://www.webperformancetoday.com/2013/06/12/11-faqs-content-delivery-networks-cdn-web-performance/
[15] http://www.feedthebot.com/pagespeed/image-dimensions.html
[16] http://www.feedthebot.com/pagespeed/keep-alive.html
[17] http://css-tricks.com/css-sprites/
[18] http://stevesouders.com/hpws/rules.php
[19] https://developers.google.com/speed/docs/insights/EnableCompression
[20] http://en.wikipedia.org/wiki/HTTP_compression
[21] https://developer.yahoo.com/performance/rules.html#minify
[22] http://gtmetrix.com/specify-a-character-set-early.html
[23] https://developer.yahoo.com/performance/rules.html#js_bottom
[24] http://www.stevesouders.com/blog/2013/11/07/prebrowsing/
[25] http://metadataconsulting.blogspot.ca/2013/03/browser-max-parallel-resource-requests.html

[26] http://en.wikipedia.org/wiki/Content_delivery_network
[27] http://www.feedthebot.com/pagespeed/enable-compression.html
[28] http://en.wikipedia.org/wiki/Minification_%28programming%29
[29] http://crockford.com/javascript/jsmin
[30] http://javascript-minifier.com/
[31] http://cssminifier.com/
[32] http://en.wikipedia.org/wiki/Google_%28verb%29
[33] https://developer.yahoo.com/performance/rules.html
[34] https://developers.google.com/speed/docs/insights/rules?csw=1

About The Author

Vijay Pallithekethil is a Software Tester with Professional Quality Assurance and has 9 years of experience providing performance testing solutions and working in test automation. He is knowledgeable in performance testing techniques and how it can be integrated as a part of the software development process.