Keys to Web Front End Performance Optimization
Contents
Preface ................................................................................................................................. 3
Web Front End Performance Paradigm .............................................................................. 4
Best practices/Optimizations enhancing the Web Front End Performance ....................... 5
“WWW” of Performance Monitoring .................................................................................. 8
Conclusions & End Notes .................................................................................................. 8
Preface

With the evolution of the internet, the complexity of web applications has increased extensively with more scalability and enhancements in the coding frameworks, computing resources. At the same time there was evolvement in web front end frameworks and end user interface. Apparently the end user expectations on the web performance and perception of the speed changed tremendously. Page response time has been directly linked with revenues, with multiple reports from the organizations and service providers underlining this fact.

There is a misconception that web application performance can be improved by optimizing server side response only. In fact, if entire user experience is considered the 60-75% of page load time is spent on the client side. This load time can be optimized by just focusing on front end of the application as compared to the 30-40% of server side optimization. Also front end and back end optimization differ with former being carried out from single user perspective and later one with multiple user perspective (with concurrency). Both areas are not mutually exclusive and complement each other.

This article focuses on the front end optimization techniques and measurement tools for the same.
Web Front End Performance Paradigm

Most of the performance teams are aware of the endless back end performance indicators (KPI) but not with the same matrix for front end. The web front end too has performance matrix or KPI as famously known against which the performance is evaluated.

- Time to First Byte (TTFB)
- First Impression Time
- No of HTTP Requests
- No of XHR
- Page Redirects
- 40X Errors
- CSS Performance
- JS Performance
- Blocking scripts
- OnLoad Time
- Total Load Time
- Time on Network
- Connect Time
- Transfer Time
- DNS Lookup Time
- Size of the response
- Minification of the resources (HTML,CSS,JS)
- Compression of the resources (gzip)
- Performance baselines against different versions of browsers

Typical waterfall chart for page load looks like:
This is the timeline chart that most front end performance tools would give. For illustrative purpose the example of e-commerce site snapdeal.com is considered. In the timeline chart we can see, the time required for server to create HTML response and send to client is only 440 ms and rest of the time is spent in getting embedded resources and rendering activities.

**Best practices/Optimizations enhancing the Web Front End Performance**

**Reduce no of HTTP Requests**

Every HTTP request is associated with round trip which also involves the network latency. The more the no of requests made to server to get all resources of web page, more will be round trips and hence by default more response and page render time. The latency between HTTP requests, and also the limitation in web browsers in regards to how many parallel HTTP requests a web browser can make, vastly affects the loading time of a web page.

**Solutions:**

- Combine all resources.
- Use Base64 data URI as background image:

The base64 is encoding scheme that translates binary data into the ASCII string format. In most of the cases we observe the image files in CSS are referenced through data URI which triggers HTTP request. More the no of such references, more will be count of HTTP requests. With base64 data URI instead of adding external reference to the image, the image data is embedded in the CSS thus completely eliminating the HTTP request. For sure, this would increase the size of the file but this can be overcome again by doing compression. Also there is gain on the network latency involved. This technique can serve huge gain for mobile sites where each HTTP request over cellular network comes with higher price tag on latency per request/response.
Compression of Resources:

The compression of resource addresses the size of the data transmitted over the network. The resource files like HTML, CSS, and JS can be compressed and then transmitted over. This reduces the download time for browser enhancing page load time. This compression always comes with additional CPU cycles and hence this optimization has to be measured first before applying. Also, it is better to add compression settings at Web Server layer rather on application server layer as additional CPU cycle overhead can be transferred to web server and utilize same on application server layer for better computations. Compression can be achieved by performing the configuring changes at server level like ‘mod_deflate’ module in case Apache web server.

JS & CSS Placement in Page

The problem with the scripts is they block parallel downloads. The JS should be placed at the bottom and CSS resource files at the top of the page. This avoids delayed downloads and blank white screen. Placing the scripts at the bottom would help to download resources faster as it won’t be blocked for execution of JS. Although this can not be implemented 100% as there will be some JS in the HEAD of the DOM, it can be implemented for other JS which can be executed later.

Putting Stylesheets at the top makes page to load progressively and hence appears to load faster. If the Stylesheets are placed at the bottom, browsers block rendering of the page to avoid redrawing elements if their style changes. Ultimately user is blocked at blank white screen.

Avoid Redirects

Redirects slow down the end user response. Inserting the redirect between user and HTML document delays everything, none of the resource can be downloaded in this period. Page can’t be rendered until the HTML is received by the client.

The most common use of redirect is to connect old website to new website. Using redirect to connect two websites is simple and required less coding efforts but degrades the user experience and delays the page load times. Another commonly observed redirect is in the case where trailing slash is missing from URL. For e.g. http://testsite.domain1.com/test2 would result in redirect to http://testsite.domain1.com/test2/. These both cases can be handled elegantly with some come configuration at web server. For apache web server this can be done in mod_rewrite module.

The HTTP redirects trigger additional HTTP request-response and add round-trip time latency. Few additional pointers to avoid redirects

- Don’t reference the URLs in the page that are known to redirect to other URLs.
- Never use multiple redirects to get required resources.
Avoid Bad Requests (40X)

Removing the broken links or requests for non-existing resources reduces the no of the HTTP requests to application server and hence reduction in round-trip latency time. These requests are typically observed when resources are moved and deleted or there are changes in the code. These requests are absolutely not required and can be easily removed within functional testing with the help of various tools like firebug or developer tools in the chrome.

CSS Sprites

Downloading the multiple images increases no of HTTP requests, no of bytes downloaded by web page and adds multiple round-trips. This delays downloading of the resources and increases the page load time. The multiple images can be combined using CSS sprites which would reduce the HTTP requests and avoid round-trips. Spriting the images reduces the request overhead associated with every request. This makes small size images the ideal candidate for CSS sprites where request overhead i.e. round trip would be more significant than actual transport time on network. The images that are loaded together also can be combined using CSS sprites. For e.g. Home, Back & Forward links on the webpage can be combined like below:

![CSS Sprite](image1.png)

With the help of the CSS, any required image can be displayed.

Minify Resource

Stylesheets and JS files do contain the whitespaces or commented code, developer comments. This part increases the file size and is not useful for production code. Hence this can be removed to reduce the file size. The minified resources can be further compressed to get least possible size for resource files. To There is concern associated with this approach as it becomes difficult to debug the JS code at production when resources are minified. Hence it is best to test the application with un-minified resources, fix bugs if any and then deploy minified resources. There are different algorithms/strategies available for minification extent like remove only whitespaces, optimize variable names, remove commented code or combination of all. The minification and compression optimizations have very high probability to provide performance gain.

Predictive Performance Optimization

The predictive performance optimization is more related with the perception of the speed by the end user and less with actual gain in performance which is quantified by improvement in the response time
or other computing resources. Predictive as name suggests is to predict the highly probable actions of the end user and prepare the response before end user actually performs that action. This is achieved by using two tags in DOM – ‘prefetch’ & ‘preload’. Pre-fetch is like to instruct the client to download certain resources which would be required for future actions performed by the user. Pre-load is like to instruct the client to load the page in the background without obstructing the present interface of the end user. So when user navigates to the page which was assigned with preload tag, it would be displayed as if page is fetched from cache. Here in reality the page is not fetched from the cache but requested from the server but the time where end user is interacting with current interface is utilized to load or fetch resources required for highly probable future action.

“WWW” of Performance Monitoring
Every application performance monitoring must be set up so as to address the ‘WWW’ i.e. What, Why, When. Either the over or under application monitoring prove to be not useful since it generates too much of data which is difficult to analyze, with utilizing more & more resources and simultaneously adding unnecessary overhead on the system, or too less data to draw conclusions and pin-point bottlenecks. There is plethora of tools for front end performance measurement; few noteworthy of them are-

• YSlow
• PageSpeed
• HTTPPerf
• Firebug
• WebPageTest
• Dynatrace

mod_pagespeed
The ‘mod_pagespeed’ is the apache HTTP server module which automatically applies best practices like minification, compression to pages and associated assets i.e. images, CSS or JS without modifying the existing content²

Conclusions & End Notes

The web front end performance is equally important with back end performance. The front end performance can be improved by less effort compared to that required for back end and without digging much into code. It is important to understand that performance optimization is not ‘one size fits all’ solution. The optimization techniques and their outcome would vary with different applications, design patterns, visitor types and devices. There is common saying in the performance engineering world - “Premature optimization is the root of all evil” i.e. it is important to measure before actual optimizations. It is important to apply the optimization techniques and then test to quantify their
outcome which enables informed optimization decision making process and that’s why first step to start with is ‘Test & Measure’.

References

1. http://www.w3schools.com/css/css_imageSprites.asp