A Case Study for Improving the Performance of Web Application

Balamurugan Subrayen¹, Gurumoorthi Elangovan², Vasuki Muthusamy³, Angayarkanni Anantharajan⁴
Department of MCA, Sri Manakula Vinayager Engg.College.Pondicherry, India
Email:chella40978@gmail.com, egurumoonthymca@gmail.com, dheshna@gmail.com, soundaraj07@gmail.com

Abstract- In the IT world, software applications are being rapidly developed. Clients, and so employers, are just looking for those teams/individuals who can build up applications rapidly, just bothering to make their application live; but what often happens after an application goes live is that users start to use the application and it doesn’t respond well. At this point, clients start to lose users and business. To code an application is not a big agreement; I believe it can be done by virtually anyone, meaning it is not necessary to have great knowledge or experience. Improving performance of an existing application (especially a one put together rapidly) could be quite risky and could cause many current issues. Things must be planned first to avoid horrible results.

Keywords-Performance,Web application, Improving, Software application, Optimizing.

I. INTRODUCTION

Web applications are an integral component of any and every website, especially, a business website. These web enabled applications streamline a business process and even go onto improves business performance. The best web applications are known for attracting a huge amount of traffic, which is always good news for a website. But, there are times when even the best of applications don’t perform along expected lines. They fall short of what the users expect from them, which can ultimately lead to them being replaced or reengineered. The website performance mainly focuses on the techniques and tools that can be use to improve a website’s performance. Optimizing web application performance is all about numbers and metrics so, before delve into optimization techniques, it is essential to understand what can be optimized and how to measure improvements in performance. In this post, we will review the five areas where website performance can be improved, how to establish a performance baseline, and how to measure progress. The overall goal of improving performance is to minimize the perceived delay the user experiences between the moment he clicks on the link and the page is finally displayed. The reason why we are focusing on minimizing the user-perceived delay rather than any other metric is because, in the end, what matters is improving your user’s/visitor’s experience. Having a user-centric mind set is important because it gives us a clear way to prioritize where to spend our resources and focus our efforts. For example, even if implementing a new caching system seems exciting, it might not be useful if the resource-processing time only accounts for 5% of the user-perceived delay. Of course, if your servers are under heavy load reducing it is important, but I believe that only looking at performance from the user’s point of view will give you the whole picture.

II. ANALYZING WEB APPLICATION PERFORMANCE

Even if a web page appears to the users as a single entity with a define URL, behind the scene it is assemblage of web resources that the browser fetches and renders. Even the smallest page is composed of multiples resources that usually include images, JavaScript code, and cascading style sheet (CSS). Before the user sees the page and can interact with it, at least some of these resources must be fetched and processed by the browser. For this reason, we have four non-mutually exclusive strategies to reduce the perceived delay: Reducing the time the browser takes to fetch a given resource. This can be done, for instance, by reducing the server processing time, using browser caching, and HTTP pipelining. Decreasing the number of requests. This can be done by reducing the number of resources by using sprites and combining JavaScript. To decrease the number of elements that need to be effectively loaded, we can leverage the browser cache and use a mutualized version of popular JavaScript libraries (i.e., JQuery). Optimizing the rendering speed. This can be done, for example, by using more efficient CSS selectors, a better page layout, and optimizing the JavaScript code. Making the rendering time appear shorter. Leverage how humans perceive information to make the delay appear less than it really is by adding loading indicators, pre-caching, and deferred loading strategies (the famous AJAX paradigm).

III. LIFE CYCLE OF WEB RESOURCE

For each resource the browser fetches, the five steps depicted in the following diagram occur, caching mechanisms here as they will be the subjects of an entire post. In the first step, resolving, the browser needs to translate the URL into an IP address by performing DNS queries. Believe it or not, this step is not free, as it can take a couple of hundred milliseconds, depending of your DNS server and visitor physical locations, as we will see below. In a requesting step, the browser connects to your web server to ask for the resource. The duration of this step does not depend on your server bandwidth but on the physical distance between your web server and your visitor. If your server is located in Europe and your visitor on the east coast of the United States, no matter what you do, the connecting step will takes at least 60 ms because of the speed of light. As we will discuss extensively in upcoming blog posts, using a CDN (content delivery network) will help reduce this delay by
relocating your content closer to your client. In the processing step, the server generates the content or reads the resources from the disk. The duration of this step can be reduced by implementing server side caching, using more powerful servers, and optimizing your web application code. Load balancing between servers is also one of the optimization techniques that can be applied at this stage.

The duration of the transferring step is mainly dominated by how big the element is and how much bandwidth the clients have with your server. Having smaller elements and compressing them can reduce the duration of this step. For example dedicated software exists to make JavaScript and CSS files smaller (this process is called minification). The download speed can be improved by having a better bandwidth provider, better peering, or using CDN. The rendering step occurs on the browser side. The rendering speed can be improved by making the browser’s job easier. This includes adding dimension to the images, writing faster JavaScript code and linear zing the page layout.

IV. MEASURE WEB APPLICATION PERFORMANCE

We need the right tools to measure our progress and lot of performance/benchmark tools, all of which with their utility. For example recommendation tools, such as yslow, that analyzes your page and gives you recommendations on how to improve your performance are very useful and will be covered in an upcoming post. However, to start optimizing your website, you only need two kinds of tools: A browser performance monitor and a resource performance monitor. Here is the short list of those I use daily. I am sure there are a ton of others tools that are worth mentioning, if you know one, let me know by commenting or tweeting browser performance monitor is the essential tool that will allow you to understand how the browser spends time rendering your page. Every major browser has either a built-in monitor (Chrome, Safari, and Internet Explorer 8/9) or an add-on that provides it (Firebug for Firefox). Analyzing resources loading time with a browser is useful to get a quick estimate of the performance, but be aware that it won’t give you the full picture, for three reasons. First, it only tells you how fast it is for you. For visitors from a different country and ISP, these numbers will be different in term of latency and download speed. Second, the browser does not tell you how long it took to resolve the URL. While in some cases (e.g., browser DNS-prefetching) it does not matter, it is still important to know this. Last, but not least, you don’t know how stable these numbers are, so you need multiple data points. These data will be provided by any good monitoring service.

V. IMPROVE THE PERFORMANCE

The following are a few points that can make a site scalable and reliable; but which may initially slow down development. I believe that overall, when maintenance and future changes are taken into account, total development time would be reduced.

1. Minimize HTTP based Requests

Serving images - no matter if they are of less than 1 KB - as separate web resources, cause separate web requests to the server, which impact performance.

**Solutions:**
Use Image Maps to merge up images, though image Maps could only merge up those images which are in sequence, like navigation images, so it depends upon your web site/page design. Use Inline images. Inline images could increase your HTML page size but would cause fewer requests to the server. CSS Sprites can also be used to merge up images and setting their position and backgrounds. Using CSS is very good practice but serving style sheets as separate resources, thus causing separate requests, should be considered very carefully.

**Solutions:**
Try your best to combine all your CSS based classes into a single .css file as lot of .css files will cause a large amount of requests, regardless of the file sizes. css files are normally cached by browsers, so a single and heavy .css file doesn’t cause a long wait on each page request. Inline .css classes could make HTML heavy, so again: go ahead with a single.css file. JavaScript is an awesome scripting language which can be quite powerful to play with. Nonetheless, it should be used carefully not only for request size issues; but also because it can have a way of causing unpredictable performance issues. Inline JavaScript could make the HTML page heavy, so it’s preferred to serve separate .js files or a single JavaScript file to keep all JavaScript-based scripts in a single place. JavaScript files also get cached automatically by browsers, so they usually aren’t requested each time the page is loaded by the browsers.

2. HTTP Compression

HTTP Compression is used to compress contents from the web server. HTTP requests and responses could be compressed, which can result in great performance gains. Through HTTP compression, the size of the payload can be reduced by about 50%, which is great. Isn’t it? HTTP Compression is now widely supported by browsers and web servers. If HTTP compression is enabled on the web server, and if the request header includes an Accept-Encoding: gzip, deflate header, the browser supports gzip and deflate compression mechanisms, so the response can be
compressed in any of the given formats by the web server in order to reduce the payload size. This leads to an increase in performance. Latter that compressed response is decompressed by the browser and rendered normally.

3. Correct Formatted Images at the Right Place
Normally designers use JPG or GIF formats quite randomly and ignore some other good formats to compress images.

**Solution:** Correct format should be used for right purpose like If you have to place a background image, some large image or a screenshot then the suggested format is JPG/JPEG. If you have to use small graphics like button images, header images, footer images, navigation bar images or clip arts, then the suggested format is PNG. If an image is not required to be in high or true colors and 256 colors are enough, then GIF is preferred.

4. Compress CSS, JavaScript and Images
CSS files (.css), images and JavaScript (.js) files can be compressed, as normally .css and .js files contain unnecessary spaces, comments, unnecessary code and such other things. A number of high quality (and free) utilities are available to help you pre-compress your files. I have used these utilities and seen compression results of about 50% in file size reduction after using such loss-less compression, so I recommend them.

5. CSS at Top
The recommended approach is to put CSS links on top of the web page, as it makes the page render progressively efficient. Since users want to see the contents of a page whilst it’s loading rather than white spaces, contents/formats should be given on top. HTML Specifications clearly say to declare style sheets in the head section of a web page.

6. Javascript at Bottom
When scripts are defined on top of the page they can take unnecessary time to load; they don’t show the contents that users are expecting after making any request to an HTTP web server. It’s better to display a the HTML contents of a page, then load any scripting code (when possible, of course). Preferably use/link up JavaScript-based scripts at the bottom of a web page. Alternatively you can use the defer attribute, which runs the script at the end of page loading, but that is not the preferable approach as it is not browser independent. For example, Firefox doesn’t support it and could mess up with document. Write, so only use it once you fully understand the implications.

7. Content Delivery Network: (CDN)
When a browser makes a request to any web page – that is, he types a URL/URI of any web page or web site, a request goes through many hops (routers and computers) and then finally reaches its final destination. This happens both for requests and responses. This operation affects performance and can severely effect load time. A Content Delivery Network implies a collection of computers, distributed all over the world, which deliver data (contents). Through a CDN you can have your website data on multiple servers distributed in different locations around the world. Distribute web application data in different places around the world so request can be served from the nearest location and save time (which means performance and money as well).

8. Ajax
Ajax is being increasingly used to improve usability, but oftentimes in a way which increases overall server load.

**Solutions:**
Preferably use the GET method for Ajax based Requests, because if you use POST method then the request header would be sent first, followed by the data, which basically splits the request in two steps. A single-step request can be achieved with GET if a cookie is not too long and the URL is not larger than 2k. When using ASP.NET AJAX and the UpdatePanel control for partial page rendering, use the maximum number of update panels to update small chunks of page, but use them wisely. Don’t set the Update property to Always unless needed. Instead, set the update mode to Conditional, otherwise all the partial chunks would be sent together after each asynchronous postback. Ajax based requests can also be cached when using the GET method. If the URL is the same, then cached data can be used from the client, and a round trip to the server can be avoided.

9. Ajax vs. Callback
Ajax is a great solution for asynchronous communication between client (web browser) and HTTP servers, but one solution can’t be applied to every problem. This means that Ajax is great mechanism for sending requests to the server without making a full page postback, but what if you need to send a request to the server and don’t even need partial rendering?

**Solution:** best solution is Callback.
For example, if you need to check whether a user exists or not, or if a user has forgotten his/her password and you just need to send a request to the server to check if user name exist, there is no need for client-side render - just a server side operation.

10. Reduce Cookie size
Cookies are stored on the client side to keep information about users (authentication and personalization). Since HTTP is a stateless protocol, cookies are common in web development to maintain information and state. Cookies are sent with every HTTP requests, so try to keep them low in size to minimize effects on the HTTP response. Cookie’s size should be minimized as much as possible. Cookies shouldn’t contain secret information. If really needed, that information should be either encrypted or encoded. Try to minimize the number of cookies by removing unnecessary cookies. Cookies should expire as soon as they become useless for an application.

11. Use Cache appropriately
Cache mechanism is a great way to save server round trips - and also database server round trips - as both round trips are expensive processes. By caching data we can avoid hitting them when unnecessary. Following are few guidelines for implementing caching;

**Solutions:**
Static contents should be cached, like —Contact usl and —About usl pages, and such other pages which contain static information. If a page is not fully static, it contains some dynamic information. Such pages can leverage the ASP.NET technology, which supports partial page caching. If
data is dynamically accessed and used in web pages - like data is being accessed from some file or database - and even if data is consistently or regularly changed, then that data could be cached by using ASP.NET 2.0 cache dependency features. As soon as data changes from the back-end by some other means, the cache would be updated. Now that web technologies such ASP.NET has matured and offers such great caching capabilities, there's really no reason not to make extensive use of them.

12. Upload compiled code rather than source code
Pre-compiled ASP.NET pages perform much better than source code versions. Actually pre-compilation give web sites a performance boost especially when the first request is made to a folder containing that resource. Uploading a pre-compiled version boosts up performance since the server doesn’t need to compile a page at request-time.

VI. CONCLUSION
In this study, we have discussed best Practices for speeding up Web Site to gain better performance which is For HTTP compression, GZip is considered the most effective and most popular by means of browsers and HTTP server. It can reduce file size up to 70% in size. Always keep JavaScript and CSS in external files. Avoid redirects until needed. Server Transfer is also provided so consider that as well since it performs better in some conditions. Minimize use of Iframes as its costly. Avoid try-catch blocks for control-flow as they perform poorly. Exceptions should be used only in truly exceptional situations. Minimize Cookie/CSS sizes. Minimize DOM objects on page as they are heavy weight. Use link tags rather than @import to use/link up CSS. Favicon, being a static image displayed in the browser’s address bar, should be cacheable and compressed. Always prefer a cache-friendly folder structure. For example, create specific folders for static contents, like /static for static images/static pages… SSL can never be cached so minimize its usage. Keep it for those pages which need to be secure, rather than using it for all the pages. HTTP Post requests can’t be cached, so choose the HTTP method appropriately. Prevent Denial of Service (Dos) attacks. Prevent SQL Injection. Prevent Cross Site Scripting (XSS).

REFERENCES

VIII WEBSITES

Mr. Balamurugan Souprayen received his Master’s Degree in Computer Applications in 2003 and Completed M.Phil (Computer Science) from Karpagam University, Tamilnadu, India. He is working as Asst.Professor in Department of MCA, Sri Manakula Vinayagar Engg. College, Tamilnadu. His areas of interest are Web Technology.