Web Frontend Optimisation Techniques

COMP6205: Web Development

Dr A. Rezazadeh
December 16
Overview

Web Frontend Performance Optimization Techniques

- Caching
- Techniques for Reducing Asset Weight
- Methods to Reduce Number of Requests
- Parallel Processing
- Achieving Optimization Using HTTP2
Google’s Rules

- Avoid bad requests
- Avoid CSS expressions
- Combine external CSS
- Combine external JavaScript
- Defer loading of JavaScript
- Enable compression
- Leverage browser caching
- Leverage proxy caching
- Minify CSS
- Minify HTML
- Minify JavaScript
- Minimize request size
- Minimize DNS lookups
- Minimize redirects

- Optimize images
- Optimize the order of styles and scripts
- Parallelize downloads across hostnames
- Put CSS in the document head
- Remove unused CSS
- Serve resources from a consistent URL
- Serve scaled images
- Serve static content from a cookieless domain
- Specify a character set early
- Specify image dimensions
- Use efficient CSS selectors
Leverage Browser Caching

- Caching is a somewhat broad term, but generally refers to the storage of web resources (HTML documents, images, and so on) in a temporary location to improve performance.

  **Pro:**
  - Most web browsers implement a cache to hold *recently* and *frequently accessed* resources.
  - Browsers are usually quite well behaved and respect the caching policy dictated by the server.

  **Con**
  - Many browsers offer an easy way for the user to remove temporary data (such as cached pages, sessions, and so on) for the sake of privacy.
  - The size of the cache tends to be quite small by default.
Conditional GETs, Cache-Control and Expires Headers

- Browsers send *Conditional GETs* to know if a local, cached copy of a resource is still valid.

- Two more headers that can control caching: *Expires and Cache-Control: max-age*
  - These headers tell the browser that this resource expires on such-and-such a date.
  - Until then, you can just use your locally cached copy.
  - The result is that the client doesn’t need to issue a conditional GET.

- *Expires* was defined in HTTP 1.0, whereas the *Cache-Control family* is new to HTTP 1.1.
  - Modern browsers understand both headers
Other Kids of Caching

- **Intermediate web proxies**, and at the gateways of large internal networks.

- **Transparent proxies** *(caches)* are used by many Internet Service Providers (ISPs)
  - The downside is that there is some increased latency for resources that are not in the intermediate’s cache.

- **Reverse proxies** sit in front of web servers, and the web server itself utilizes caching.
  - **Caching**: they can be used to lighten load on the back-end web server by serving up cached versions of dynamically generated pages (thus cutting CPU usage).
  - **Load balancing**: they can be used for load-balancing multiple back-end web servers.
DNS Caching

- DNS caching can occur at many levels
  - DNS Caching by the Browser
    - For example, Opera caches DNS answers for 10 minutes, while Internet Explorer (IE) 4 and upward cache for 30 minutes
  - Operating system level
    - using a static hosts file (such as /etc/hosts on Linux)
    - Local resolver or the resolving DNS server that is hosted by the client’s ISP
  - DNS Caching can help anything from a few milliseconds to perhaps one-half a second
DNS Prefetching

- Prefetching involves performing DNS lookups on URLs linked to in the HTML document, in anticipation that the user may eventually click one of these links.
  - DNS *prefetching* introduced in Chrome in 2008 for the first time
  - In **Chrome**, prefetching occurs in the background after the HTML document has been fetched — either while additional resources are retrieved, or the user is reading the page
  - **Firefox** uses prefetching after downloading the initial HTML file to resolving hostnames used in other resources (such as images, CSS, and JavaScript)
Reduce Asset Weight

• Use “GZIP” Compression
  – GZIP is a versatile script that can be applied to almost every asset on your website. As long as the script is called first, all files of that type (HTML, JS, CSS) on your site will be “gzipped,” i.e. compressed and less heavy.

• Minify Scripts
  – Script minification is the process of stripping out whitespace, comments, and erroneous characters from code. This makes the code more streamlined and less heavy.
Using GZIP Compression

GET /1/icons.html HTTP/1.1
Host: linuxbox.co.uk
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.1.8)
    Gecko/20100308 Iceweasel/3.5.8 (like Firefox/3.5.8) GTB7.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

HTTP/1.1 200 OK
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny8 with Suhosin-Patch
    mod_ssl/2.2.9 OpenSSL/0.9.8g mod_perl/2.0.4 Perl/v5.10.0
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
Keeping the Size Down with Minification

• *Minification* is the act to **strip out** unnecessary characters from code to reduce the size.

  – Most often, the term is applied to JavaScript, but as you shall see, the technique can also be used on CSS and (to some extent) HTML.

  – The aim of minification is, of course, to reduce file size and thus speed up transfer times for clients.

  – Using gzip compression offers bigger reductions in file size, but minification is still a useful technique, because 10% browsers that don’t support gzip compression.

• A **minifier** is the tool that does it.
The Downside with Minification

- Code becomes difficult to read and modify.
  - One solution is to store your code unminified and then minify it on-the-fly, as clients request it.
  - This is a terribly inefficient way to handle what are usually fairly static resources, and definitely not something recommended.
  - Instead, you should keep an unminified copy of the code in a separate directory and pass it through the minifier when you are ready for it to go live, possibly as part of your build process.
What are “Unnecessary Characters”?

• Whitespace and comments are the main candidates for removal.
  – Many simplistic minifiers remove only these.
  – This is often just scraping the surface of what is possible though — especially with JavaScript, which will be discussed later.

• *Be aware that minification shouldn’t be used as an excuse to write sloppy code.*
  – *Your first step should always be to manually clean up your code as much as possible.*
  – *only then should you pass it through the minifier.*
Javascript Minification

- Of the three languages (JavaScript, CSS, and HTML), JavaScript offers the most potential for minification.

  - Aside from removing whitespaces and comments, Windows-style line breaks (CRLF) can be converted to UNIX-style breaks (LF).

  - Variable names can also be shortened.
For all but the most basic websites, it is common to automate the process of deploying a new version of the site via a basic shell script or batch file. This might consist of copying the code over from a development/staging area, setting the correct file permissions, pre-compressing resources, and so on. This is an ideal place in which to perform minification.

What are “unnecessary characters”? In the languages discussed here, whitespace and comments are the main candidates for removal, and many simplistic minifiers remove only these. This is often just scraping the surface of what is possible though — especially with JavaScript, which is where the discussion in this chapter starts.

Be aware that minification shouldn’t be used as an excuse to write sloppy code. Don’t fall into the trap to think that it’s okay to write bloated, badly structured code because a minifier will clean it up for you. Your first step should always be to manually clean up your code as much as possible; only then should you pass it through the minifier.

**Javascript Minification**

Of the three languages examined here (JavaScript, CSS, and HTML), JavaScript offers the most potential for minification. Aside from removing whitespaces and comments, Windows-style line breaks (CRLF) can be converted to UNIX-style breaks (LF), and variable names can be shortened.

Let’s look at the typical minification process for a small block of code. In this case, the code handles a function to toggle the “visibility” (more accurately the display) of an element:

```javascript
function toggle(elementID) {
    if ( document.getElementById(elementID).style.display != 'none' ) {
        document.getElementById(elementID).style.display = 'none';
    } else {
        document.getElementById(elementID).style.display = '';
    }
}
```

As it stands, this function weighs in at 297 bytes. Before you run it through a minifier, you should attempt some manual optimization. Using a variable holding a reference to the element’s display would be a good start, as shown here:
NOTE
For all but the most basic websites, it is common to automate the process of deploying a new version of the site via a basic shell script or batch file. This might consist of copying the code over from a development/staging area, setting the correct file permissions, pre-compressing resources, and so on. This is an ideal place in which to perform minification.

What are “unnecessary characters”? In the languages discussed here, whitespace and comments are the main candidates for removal, and many simplistic minifiers remove only these. This is often just scraping the surface of what is possible though — especially with JavaScript, which is where the discussion in this chapter starts.

NOTE
Be aware that minification shouldn’t be used as an excuse to write sloppy code. Don’t fall into the trap to think that it’s okay to write bloated, badly structure code because a minifier will clean it up for you. Your first step should always be to manually clean up your code as much as possible; only then should you pass it through the minifier.

JAVASCRIPT MINIFICATION

Of the three languages examined here (JavaScript, CSS, and HTML), JavaScript offers the most potential for minification. Aside from removing whitespaces and comments, Windows-style line breaks (CRLF) can be converted to UNIX-style breaks (LF), and variable names can be shortened.

Let’s look at the typical minification process for a small block of code. In this case, the code handles a function to toggle the “visibility” (more accurately the display) of an element:

```javascript
function toggle(elementID) {
    if ( document.getElementById(elementID).style.display != 'none' ) {
        document.getElementById(elementID).style.display  = 'none';
    } else {
        document.getElementById(elementID).style.display = '';
    }
}
```

As it stands, this function weighs in at 297 bytes.

Before you run it through a minifier, you should attempt some manual optimization. Using a variable holding a reference to the element’s display would be a good start, as shown here:

```javascript
function toggle(elementID) {
    var el = document.getElementById(elementID);
    if ( el.style.display != 'none' ) {
        el.style.display = 'none';
    } else {
        el.style.display = '';
    }
}
```

This takes the weight down to 246 bytes.

You can simplify things a bit more by getting rid of the if/else block and using the ternary operator:

```javascript
function toggle(elementID) {
    var el = document.getElementById(elementID);
    ( el.style.display != 'none' ) ? el.style.display = 'none' : el.style.display = '';
}
```

This takes it down to 193 bytes. So far, you have preserved whitespaces, and the code is still readable.

Now that you’ve seen how to clean up this code, you can pass it through a minifier.

**YUI Compressor**

One of the most popular minifiers is Yahoo’s YUI Compressor. It’s a command-line minifier, written in Java, that can process both JavaScript and CSS. You can download it from [http://developer.yahoo.com/yui/compressor/](http://developer.yahoo.com/yui/compressor/). (Of course, you must have Java installed.) Running it is simple, as shown here:

```
$ java -jar /usr/local/bin/yuicompressor-2.3.5/build/yuicompressor-2.3.5.jar input.js > output.js
```

Now try it on the sample function introduced earlier:

```
$ java -jar /usr/local/bin/yuicompressor-2.3.5/build/yuicompressor-2.3.5.jar function.js > function_minified.js
```

```javascript
function toggle(a){var b=document.getElementById(a);if(b.style.display!="none")
{b.style.display="none"}else{b.style.display=""};
```

A few things have happened here. Unnecessary whitespaces have been removed, and the two variables have had their names shortened: `elementID` to `A`, and `el` to `B`. Because these variables exist only inside the function, it’s safe to rename them without worrying about it impacting other code.

This takes the function’s size down to 93 bytes — a significant improvement from the original 342.

Just to stress the importance of manually optimizing your code first, look at how the YUI Compressor copes with the original function before you made any changes to it:

```
function toggle(A){var B=document.getElementById(A);
if(document.getElementById(A).style.display!="none")
{document.getElementById(A).style.display="none"}
else{document.getElementById(A).style.display=""};
```

Manual optimization using a variable holding a reference to the element’s display
A more optimized Version by replacing if/else block with ternary operator.

```javascript
function toggle(elementID) {
    var el = document.getElementById(elementID);
    (el.style.display != 'none') ? el.style.display = 'none' :
        el.style.display = ''; 
}
```

Weighs in at 193 bytes.
YUI Compressor

• One of the most popular minifiers is Yahoo’s YUI Compressor.
• It’s a command-line minifier, written in Java
• It can process both JavaScript and CSS.

```javascript
function toggle(a) { var b = document.getElementById(a); if (b.style.display != "none") { b.style.display = "none" } else { b.style.display = "" }; }
```

- Unnecessary whitespaces have been removed,
- the two variables have had their names shortened: `elementID` to `A`, and `el` to `B`.
- Because these variables exist only inside the function, it’s safe to rename them without worrying about it impacting other code.

Weighs in at 93 bytes
Importance of manual Optimization

• Just to stress the importance of manual optimisation look at how the YUI Compressor copes with the original function before you made any changes to it:

```javascript
function toggle(elementID) {
  var el = document.getElementById(elementID);
  (el.style.display != 'none') ? el.style.display = 'none' :
  el.style.display = '';  
}
```

- It has still renamed the local variables and removed whitespaces, but the long-winded if/else block with references to the element’s display are still there.
- The YUI Compressor may be clever, but it’s not a mind reader.
Google Closure

- The YUI Compressor is not the only JavaScript minifier out there.
- The new kid on the block at the moment is Google’s Closure Compiler (http://code.google.com/closure/compiler/),
- It supports two modes: **Standard** and **Advance** modes
- Standard mode offers levels of minification similar to the YUI Compressor
- The advanced option has the capability to make more savings.
  - Two of the most exciting features of this advanced mode are function inlining and the removal of unused code.
CSS & HTML Minifiers

- Many of the minification techniques outlined for JavaScript minification are also applicable to CSS
- Minification of CSS and JavaScript are both popular techniques. However minification of HTML has so far failed to catch on.
  - Part of the reason for this is that it requires more thought to implement.
  - CSS and JavaScript are usually static files,
  - HTML is often generated on-the-fly by assembling fragments of markup using a back-end scripting language.
Reduce Asset Weight – Image Compression

• Use **Lossy** and **Lossless** image compression
  
  – All images on a site should be compressed.
  
  – **Lossless** compression can slightly reduce the weight of an image without reducing the display quality at all.
  
  – **Lossy** image compression slightly reduces the display quality (though usually so little it’s not noticeable to the visitor) but can reduce the weight of the image on the order of 50 to 90% depending on the image and degree of loss.
Web Optimisation - Reduce Number of Requests

• Combine Scripts

  – Combining or “concatenating” files and scripts is accomplished by writing a line of code that encompasses information that was previously contained in two or more lines of code.

  – This is a common practice that can be accomplished by a developer or using free script-based tools.
Reducing the Number of HTTP Requests

• Although you may have merged your CSS and JavaScript files, there will typically still be numerous images. We can achieve further optimisation by

1. Combine images with “Data: URIs”
   – A data: URI is a method of including small images or other bits of data “inline” into an HTML file.
   – That way, instead of loading as a standalone request, the image will be downloaded along with the HTML file that includes the basic framework of the page.

2. Reducing the number of images requested using Image Maps.
Reducing the Number of HTTP Requests

3. Combine images with **CSS Sprites**
   - A sprite is a single image that contains the visual information of multiple images.
   - Spriting is done by combining each smaller individual image into one big image, and then displaying a small section of that image (a **sprite**) at defined X and Y coordinates.
   - This allows the developer to make a single request for multiple images.
Image Maps

- **Image maps** are a popular technique for mapping specific parts of the image to different destinations.

- Consider that you have 6 separate Icons like this:

- Each linked to a separate backend script like this:

  ```html
  <a href="/games.php"><img src="applications-games.png"></a>
  <a href="/videos.php"><img src="applications-multimedia.png"></a>
  <a href="/mail.php"><img src="internet-mail.png"></a>
  <a href="/calendar.php"><img src="office-calendar.png"></a>
  <a href="/search.php"><img src="system-search.png"></a>
  <a href="/help.php"><img src="help-browser.png"></a>
  ```
Image Maps – Cont.

• Because they sit adjacent on the page, these icons are ideal candidates for an image map.

• First, you combine them into a single image using GIMP.

• Next, you create the image map, as shown here:

```
<map name="navigation">
<area href="/games.php" shape="rect" coords="0,0,42,32">
<area href="/videos.php" shape="rect" coords="42,0,79,32">
<area href="/mail.php" shape="rect" coords="79,0,116,32">
<area href="/calendar.php" shape="rect" coords="117,0,154,32">
<area href="/search.php" shape="rect" coords="155,0,190,32">
<area href="/help.php" shape="rect" coords="191,0,230,32">
</map>

<img border=0 src="imagemap_nav.png" usemap="#navigation">
```
Image Maps – Cont.

- Waterfall view of the original page loading - loading time of 1.16 seconds.

- Here, the number of resources has dropped to two, and the loading time is down to 0.62 seconds.
Image Maps – Downsides

- Navigation breaks for users on text-based browsers (or those who have images turned off),

- May cause accessibility issues for text-to-speech engines.

- No interactivity is possible. You can’t have separate mouseovers for individual links.

- Images must be adjacent.
A Better Way - CSS Sprites

- The CSS background-image property enables you to set an image to be used as the background of a given element.
- One of the neat things about CSS backgrounds is that an offset for the image may be given using the background-position property.

Figure 5-14 shows an image that measures 554 pixels wide and 394 pixels high. To display this as a background image, you can use the following code:

```css
.style {
  background-image: url('/images/flowers.png');
  width: 554px;
  height: 394px;
}
```

However, using background-position, you can display only a section of the image. Now pick an arbitrary region inside the image, as shown in Figure 5-15.

The upper-left coordinates of the highlighted area are 193 pixels and 100 pixels, and the region is 95 pixels wide and 71 pixels deep. Thus, consider the following:
What Are CSS Sprites?

- CSS sprites are a way to use one large image to display multiple smaller images.
- This is used to display menus, icons or buttons in one big image while showing only a part of that image at a time and switch to another part of the image when we need it.

Apple uses CSS sprites for various states of its main navigation menu.
Why Use CSS Sprites?

- The idea behind CSS sprites originally was to simply speed up your web pages load time.
- It does this by minimizing the amount of HTTP requests that are made while your page is loading.
  - For example, if you have 10 images that can combined into 1 CSS sprite, then you will only have 1 HTTP request instead of 10.
  - Before CSS sprites were commonly used, the exact opposite was being done.
How to Create CSS Sprites?

1. Create one big image containing all the needed graphics for menus or buttons

2. You get the correct image to display by specifying the coordinates of each smaller image using the CSS background-position attribute.

3. You can do this manually, but it is easier to use the CSS Sprite Generator tool at:
   - [SpriteMe.org](http://spritegen.website-performance.org/) is another popular tools for CSS sprite creation.
Parallel Processing

- Load 3rd party assets “asynchronously”
  - Social media widgets, trackers, and other dynamic assets on your site are often hosted by third parties far and wide, on servers you have no control over.
  - When those widgets have performance issues, their problems spill onto your site and drag down performance.
  - Rewriting the script so that it loads asynchronously can help mitigate blocking behavior so that the rest of your site can load around the struggling asset.
Parallel Processing - Domain Sharding

• Domain sharding is a way to overcome the limitation of browsers that only allow a handful of open connections with a server at a time.

• This has largely been solved by newer editions of browsers that allow for up to 12 open connections at a time - but not all web users are on updated browsers.

• To do this you rewrite an asset’s URL to include a made-up alias in the place of “www” that tricks the browser into thinking it’s a different server.
Taking the advantages of HTTP2

- At a high level, HTTP/2:
  - Is binary, instead of textual
  - Is fully multiplexed, instead of ordered and blocking
  - Can therefore use one connection for parallelism
  - Uses header compression to reduce overhead
  - Allows servers to “push” responses proactively into client caches
Conclusions

• Performance tuning of Web-based systems is a multi-variable and iterative process.

• A systematic planning is an effective approach.

• The tuning process involves:
  - Application review
  - System review
  - Test design
  - Data collection and analysis
  - System and application modification

• Test design is crucial to successful performance tuning.
References

1. Professional Website Performance: Optimizing the Front-End and Back-End by Peter Smith

2. High-Performance Browser Networking by Ilya Grigorik, chapter 9 - 15

3. High Performance Web Sites by Steve Souders


5. Steve Souders Website of Web Performance Optimisation
   http://www.stevesouders.com/blog/2010/05/07/wpo-web-performance-optimization/
References

- O’REILLY
  - Using WebPageTest
  - High Performance Web Sites
  - Even Faster Web Sites

Rick Viscomi, Andy Davies & Marcel Duran