Load Speed Problems of Web Resources on the Client Side: Classification and Methods of optimization

Osama Ahmad Salim Safarini

Computer Engineering Department, College of computers and Information Technology,University of Tabuk,, Tabuk 71491, Saudi Arabia

Abstract— *This article is concerned about client side issues of web resources load process related to user agents (browsers) behavior. a lot of modern problems such as improving global availability and reducing bandwidth, the main problem they address is latency: the amount of time it takes for the host server to receive, process, and deliver on a request for a page resource (images, css files, etc.). latency depends largely on how far away the user is from the server, and it's compounded by the number of resources a web page contains; current load algorithms are investigated and all known solutions with their area or efficiency are explained. We have described four main optimization methods.*

Keywords— optimization of the web page load time, client performance, client optimization approaches.

I. INTRODUCTION

If we talk about the problems related to client performance, you should immediately refer to the ambiguity approach to solving any problem that lies in this area. When creating any external module (page or part thereof) the client architect has to make a choice. Either on this page will use its own style sheet (then to speed up loading it possible to include in the final HTML-document). Or if it will be used by the general style file, then it is necessary not to forget about caching and estimate the number of regular users.

Compromises pursue architect everywhere: to unite all the files in one or divided into several independent modules? Cache whether individual resource files needed to display the page, or to incorporate them into the document itself? Which set of browsers should be maintained, and what techniques to use while? What palette size and the degree of compression for images, and how best to break a complicated pattern into several components? For which pages may use methods of conducting resource loading after loading the page itself, and which need for pre-loading?

The questions are very much, and most of them are tied to the basic knowledge of client optimization. In our paper we tried to answer and explain some of these questions. We provided four key points of the main problem areas on page load and we described all the optimization methods related to those problem areas.

II. CLIENT ARCHITECTURE AND HOW IT DIFFERS FROM THE SERVER

The importance of client architecture currently cannot be overstated, because the vast majority of issues to accelerate the loading of web resources are associated with the client part. In an effort to create a convenient, fast and cross-browser Web resource modern architect client needs to solve many problems, coherent vision of the customer with the convenience for the users and be sure to take into account, how a web-based resource (or a portal) will be developed in the future. According to last year's survey [2] conducted by Yahoo! engineers in the field of user interfaces, 95% of the time when loading a Web resource associated with the delays on the side of the end user. And only 5% are "server" component (which includes, in addition to waiting for a response, in fact, from the server, and more time on the DNS-request time to establish TCP / IP-connections and a number of other costs). That is why the optimization of the page load time is one of the top priorities.

In addition, each problematic issue, whether the use of standards to fit on the page, the combination of background images to reduce the number of requests to the server, the use of JavaScript-logic on the page should be decided primarily on the basis not only of the actual technical specifications, but also the best performance on the browser side.

Let's look at the kind of problems you may encounter when you create high-performance web resources, and how they are best addressed.

Problem 1: Poorly Written Code

Poorly written code can lead to a host of web application issues including inefficient algorithms, memory leaks and application deadlocks. Old versions of software, or integrated legacy systems can also drag performance down. Make sure your teams are using all the tools at their disposal – from automated tools like profilers to best programming practices like code reviews.

Problem 2: Unoptimized Databases

An optimized database allows for the highest levels of security and performance, while an unoptimized database brings can destroy a production application. Missing indexes slow down the performance of SQL queries causing, which can drag down an entire site. Be sure to use scripts and file statistics to check for any inefficient queries.

Problem 3: Unmanaged Growth of Data

Data systems tend to degrade over time. Developing a plan to manage and monitor data as it grows is indispensable to your web performance success. The first step is deciding who is accountable for data growth in your business. From there, your team will need to research and determine the appropriate storage for your data needs. Look at all your options, from databases to caches to more sophisticated layered storage solutions.

Problem 4: Traffic Spikes

We generally think of increased traffic as a good thing. However, anyone who has experienced major traffic spikes after a marketing promotion or viral video knows what can happen when you aren't properly prepared for them. Planning ahead is key, and set up an early warning system through simulated user monitoring systems like NeoSense. That way, you'll see when traffic is impacting transactions before your users have a bad experience.

Problem 5: Poor Load Distribution

Poor load distribution can cause slow response times by incorrectly assigning new site visitors to bogged-down servers instead of others with cycles to spare. If too many people are on the same server, they're going to experience problems, even if the overall system is well under capacity. It is imperative to test with a product like NeoLoad as it will help you find any infrastructural weaknesses at hand.

Problem 6: Default Configurations

Systems must be properly tuned. While default configurations make it easy to get new components up and running, they're not always appropriate for your web applications in a live production environment. Every setting should be checked: review thread counts, allocated memory and permissions. Confirm that all configuration parameters suit the demands placed on your web application, and aren't the way they are just out of convenience.

Problem 7: DNS, Firewall, and Network Connectivity

DNS queries make up the majority of web traffic. That's why a DNS issue can cause so much trouble, preventing visitors from accessing your site and resulting in errors, 404s and incorrect pathways. Likewise, network connectivity and firewall efficiency are crucial for access and productivity. Use DNS monitoring safeguards to pinpoint problems at hand. Also, revise switches, check VLAN tags, and distribute tasks between servers. These are just a few ways to troubleshoot these types of performance issues.

Problem 8: Troublesome Third-Party Services

If you rely on third-party services, you know that some slowdowns are out of your control. Who hasn't experienced a stalled page, waiting to load an ad from someone else's ad server. If your users are experiencing problems, it's essential to determine if the problem is on your side or that of the third-party. If you decide to continue using the third-party service, look at making some design changes to protect your site from at least some of the effects of a third-party service issue. Finally, make sure your company and the off-service provider are clear on performance guarantees.

Problem 9: Shared Resources and Virtual Machines

Just about every web application today relies on virtual machines for everything from scalability to management to system recovery. However, sometimes the way these virtual systems are organized – hundreds of VMs on a single physical server – can result in problems where one bogged-down system affects all the others. After all, contention is bound to happen. Monitor systems closely so that if one VM is causing problems, you can deal with the side-effects quickly.

Problem 10: The Domino Effect

Finally, make sure you realize that a failure in one location may affect other spots in ways you wouldn't necessarily think of. Problems compound upon themselves, making it hard to determine what is really going on. You've got to train your team to find root causes, backtracing through problems to find the real culprit. You may even want to think about mimicking Netflix's Chaos Monkey strategy, which introduces abnormal errors in the network to push the boundaries of resiliency and recovery.

2.1 Time Aspects of Loading a Web Page

The corresponding study demonstrated that user irritation is greatly increased if the page loading speed exceeds 8-10 seconds without any user notification about the boot process [3]. Recent work in this area has shown that users with broadband access are even less tolerant of delays at loading web pages than with the users with a narrower channel. The survey, conducted by Jupiter Research [4], it was found that 33% of the user-speed connections do not want to wait for more than 4 seconds when the page loads, while 43% of users do not wait more than 6 seconds.

In a study conducted in 2004, Fiona Nah found that the tolerant waiting time (TWT) for broken links (without feedback) is between 5 and 8 seconds. [5]. With the addition of notifying the user of the boot process (feedback), for example, load indicator, TWT has increased to 38 seconds. The distribution of TWT to go to retry broken links was maximum in the region of 2-3 seconds (without feedback). Nah concluded that TWT web users have a maximum of about 2 seconds. Taking into account the desire to visit the user web resource repeatedly, Dennis Galletta and others showed that the curve is smoothed at 4 seconds or more and goes to zero in the region of 8 seconds or more [6].

III. THE MAIN PROBLEM AREAS ON PAGE LOAD ANY WEB RESOURCE

The main problem areas on page load any web resource can be divided into four key points:

1. Preloading - the appearance of pages in the user's browser. After a moment of waiting at the entrance to the download web resource in the user's browser page appears drawn. At this point, likely missing page drawings and may not be fully functioning JavaScript-logic.

2. Interactive loading - appearance interactivity loaded web page. Typically, the whole logic of client interaction is available immediately after the initial page load (stage 1). however, in some cases (for them speech will be a little further) support this logic a bit delayed in time from the appearance of the main picture in the user's browser.

3. Full page loading - Page Web resource appeared fully in the browser, for it presents all the declared information, and it is ready for further user action.

4. Post-loading pages - At this stage, fully loaded page may be (in the invisible to the user mode) to carry out the loading and caching some resources or components. They may require the user when switching to other page of this web site, or to display (but not the logic functioning) of any interactive effects. For the majority of Web resources at this time is to distinguish only preload (which is enabled by default an interactive boot) and a full load of the page. Post-loading, unfortunately, it is now used very little.

Optimize the speed of loading web pages focused on two key aspects: acceleration of reload and accelerate the main load. All basic techniques are focused on this, because these two stages are perceived by the user as a "loading" web page.

3.1 The Effectiveness of the Main optimization Methods

If describe all the optimization methods, they are divided into four main groups:

1. Reducing the amount of data provided (this includes the use of compression algorithms and corresponding formats for images).

2. Caching (using paired client-server headers to reduce the transmission time information when it is maintaining its relevance).

3. Fewer resources (various methods of combining downloadable files).

4. "Visual" optimization (which includes the separation of algorithms boot process into 4 stages for maximum acceleration loading the main stages, as well as a number of methods associated with parallel streams of loading).

By reducing the amount of data will be the most efficient archiving (gzip / deflate) on the server. Practically all modern Internet industry giants are now issued in a text file gzip-format (this GoogLe, and Yahoo !, and Yandex). There are certain problems with the perception of these files on some browsers; however, almost all of them at the moment can be overcome. This approach is the easiest to use, and therefore has the greatest efficiency: minimum action leads to at least the maximum result.

Caching does not require deep knowledge of network protocols and the subtleties of imposition, but the ability (when large quantities of regular visitors) have a significant impact on the speed of loading pages specifically for them.

Further efficiency is generally used as a text association (.html, .css, .js) files and graphics used for decoration purposes. Text files combine very simple, and we can save a considerable amount of time it takes to additional requests to the server. By combining and image files are commonly used technologies CSS Sprites (Image Map), which is not easy to automate, but large numbers of icons on a page, it can greatly speed up the load.

The methods of "visual" optimization can be attributed as an extremal optimization: When all related files are included in the final - and load distribution for loading files on multiple servers. On the real-time display of the page, these actions do not affect much, in some cases, their implementation involves considerable difficulties. However, in the case of high load, even a few milliseconds can affect significant (in an absolute sense) increase profits.

The main criteria that should determine which methods and how much should be applied, of course, is the audience resource. For each Web resource can identify several specific groups of pages that are visited by one or another type of audience. The first group includes Web pages that new users visited every time. This special promotional page, whose task is to direct sales of the product. This is the page of ads that users should see one, a maximum of two times. Etc. Followers of such pages by 99.9% consists of new users. Therefore, for them it is necessary to apply methods that reduce, first of all, number of appeals to the server to display the page: combining files and extreme optimization.

The second group can be attributed pages audience which often varies; however, part of it can view content an unlimited number of times. For these pages, you can select the characteristic core of regular visitors; however, it is not more than 30-40% of the total. Most web resources that "live" in the search traffic, is a remarkable example, fully correspond to this group. For these pages, first and foremost, is to consider methods of reducing the number of requests (CSS Sprites), the possible minimization of all the text files (HTML, CSS, JavaScript). However, the use of caching is justified in this case to a lesser extent as to reduce the load time of the page is not so much (if taking a weighted average) than, e.g., parallel queries. Finally, the third group includes all the remaining pages, namely those which the audience is more constant (a specific number of parameters business efficiency of various audience groups should be considered, however, the characteristic values are - that's 30% of regular users of the resource). In this group, the most effective will, of course, caching and optimizing the speed of the JavaScript and Flash-animation - in fact it will "eat" most of the time.

3.2 Compression Methods

The main tools for reducing the amount of data are various minimizers and Obfuscators (for JavaScript-files), archiving, and also a number of utilities to reduce the size of images. Let's take them in order. As shown by the testing means compressing CSS, best copes with this task the project CSS Tidy [8] (about the same level with him is YUI Compressor [9]), which, together with additional archiving files, you can get a win up to 85% [10] .

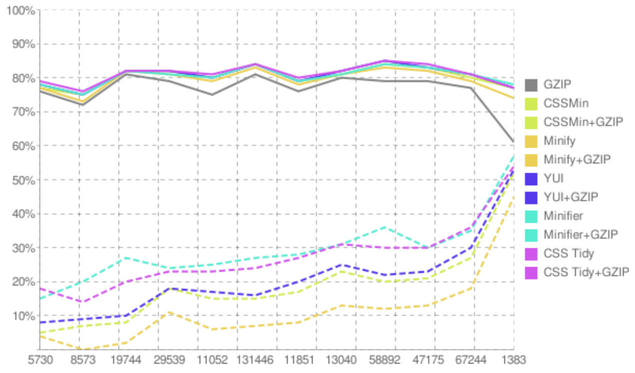


Fig. 1: Dependence winnings to compress CSS-files using a variety of tools and archiving

For the JavaScript-files, the situation is somewhat more interesting. [11]. If you apply the archive, it is best to use the YUI Compressor [9], as it is, on average, in this case, compresses better. If archiving for JavaScript-files cannot be used, the leader in compression is Dean Edwards Packer [12]; however, it introduces extra costs for his "decompression". Studies have shown that for users who will be mainly load JavaScript from the cache, you should use compression without obfuscation.

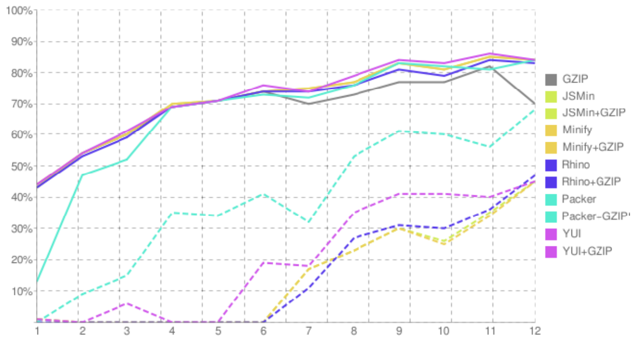
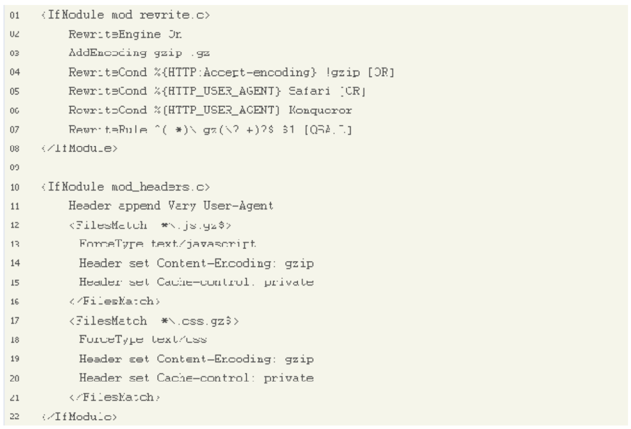


Fig. 2:Dependence winnings for compressing JavaScript-files using a variety of tools and archiving

Using archiving through mod_gzip or mod_deflate for a web server by Apache (and corresponding modules for other web servers) can significantly reduce the size of uploaded files. However, in case of very rapid channel users (e.g., local resource), and the limited resources of the server (high specific load on the creation of the page) would be wiser not to use compression [13]. Also worth for archived files to add the appropriate headers (Cache-Control: private), to avoid a number of problems with local caching proxy server. To archive CSS- and JavaScript-files is also necessary to exclude Safari (for Windows-based platforms) and Konqueror from those browsers that can send gzip-files: these browsers until recently, experts were not able to recognize them correctly.



```
<IfModule mod_rewrite.c>
    RewriteEngine On
    AddEncoding gzip .gz
    RewriteCond %{HTTP:Accept-encoding} !gzip [OR]
    RewriteCond %{HTTP_USER_AGENT} Safari [OR]
    RewriteCond %{HTTP_USER_AGENT} Konqueror
    RewriteRule ^(.*)\.gz(\?.+)?$ $1 [QSA,L]
</IfModule>

<IfModule mod_headers.c>
    Header append Vary User-Agent
    <FilesMatch .*\.js.gz$>
     ForceType text/javascript
     Header set Content-Encoding: gzip
     Header set Cache-control: private
    </FilesMatch>
    <FilesMatch .*\.css.gz$>
     ForceType text/css
     Header set Content-Encoding: gzip
     Header set Cache-control: private
    </FilesMatch>
</IfModule>
```

Fig. 3: Example configuration of Apache web server to enable compression of CSS-and JavaScript-files

For most graphic elements it is recommended to use the format of PNG, as it is more economical than GIF [14] in the diagrams and drawings with a limited color palette.

However, for small images GIF-format can be better. Currently PNG-images are supported, practically by all browsers.

Now there is a problem with alpha channel in Internet Explorer (which promise to correct in version 8), however, version 6 and 7 it can be solved through the filter ImageAlphaLoader, which allows to use translucent, practically in full size. In case of problems with the coincidence of color (again, in Internet Explorer) is recommended to remove from the image gAMA-Chunk responsible for transparency (in this case, to obtain a transparent image with matching flowers in Internet Explorer does not work). This can also help a number of utilities that reduce the size of PNG-images: for example, pngcrush. To reduce the size of JPEG image-jpegtran utility can be applied, which does not affect the color image data, and removes comments and other meta-data.

For animated images is to use either the GIF-image with multiple frames, or DHTML-animation (using JavaScript0 logic to change PNG- or JPEG-images). Animated PNG-images are not supported by browsers [14].

After using all of the methods the final page size can be reduced by 30-80%. However, if the Web resource is loaded more than 10 seconds, this may not be enough.

3.3 Methods For Combining Files

Each request from a user's browser to the server is rather resource-intensive operation. Even if you have an open connection to the server browser still has to pass (and get) the relevant FTP- or HTTP-headers (which only increase the actual amount of transmitted information). In addition, HTTP / 1.1 specification browser does not have the right to open more than 4 connections to one server (in the past this was due to a heavy load on the server when a large number of concurrent requests, however, at present the browsers, of course, increase this number to 6 -10). Therefore, each request must wait for the next stage in its general stream, before it is processed and transmitted to the server.

With a significant number of files that are required to display the page on the web-site, this can result in a significant amount of time waiting for the user. For consideration of methods to accelerate the process in this area should refer to the above stages of loading and to focus on the possible transfer of files from the first stage (preload) to the second, third or fourth stage.

Unfortunately, for correct display of the browser page it is necessary to load all the files of styles (CSS), which are listed on it. To speed up the process of preloading is necessary to combine all the files (for different devices can be combined using the selectormedia) or even (with a small amount of their fickle audience or web resource) to place them directly in the HTML-file. In addition, the call of CSS-file on the page should be before the call to any other files (for example, favicon.ico).

The survey showed [15], to accelerate the loading HTML- and JavaScript-files optimally will unite them in a single file (to establish multiple connections consumed too many resources compared to the speed of information transfer). The most popular technique for combining images is CSS Sprites [16] (or CSS Image Map), when the display of several (tens or even hundreds) of images using one resource file. It gives a considerable gain in loading speed when using animation effects (for example, changing an image when you hover the mouse) as well as with a large number of icons on the page. When using this technique, the background image is positioned using style rules that are actually "cut" him from the general resource file.

The main recommendations to create resource files for CSS Sprites are the following:

1. Breaking all the images on 5 major groups (not recommended for use in an image file of more than 2 groups):
   1.1. Images, repeated in all directions (corresponding CSS-rule repeat).
   1.2. Images, repeated horizontally (usually repeat-x).
   1.3. Images repeated vertically (typically repeat-y).
   1.4. Images that are not repeated (typically no-repeat).
   1.5. And animated images.
2. The file size must be no more than 10-20 KB
3. There should be close association of the color image If you have a page displays a lot of small images, you may want to use the technique Image Map, in order to reduce their number.

3.4 Caching

The main technology to accelerate loading pages for regular visitors is caching, which can reduce the number of requests to the server to display the page to a minimum (ideally to zero). It is worth remembering correctly configured header Cache-Control. To clear the cache, you can always use an additional parameter in the GET-request to a resource that will cause the server to take the same physical file, and the client browser request the file and save it under a new name. For static resources is quite a long time to expose the cache (it is possible to experiment with the values of the month), for other files, this value must be equal to the average time change or absent (for HTML-files, for example).

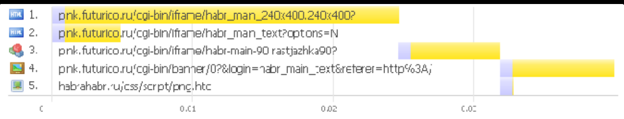

1. pnk.futurico.ru/cgi-bin/iframe/habr_man_240x400.240x400?
2. pnk.futurico.ru/cgi-bin/iframe/habr_man_text?options=N
3. pnk.futurico.ru/cgi-bin/iframe/habr-main-90 rastjazhka90?
4. pnk.futurico.ru/cgi-bin/banner/0?&login=habr_main_text&referer=http%3A/
5. habrahabr.ru/css/script/png.htc

Fig. 4:Adding caching headers for static files is capable of ten times to reduce the number of requests when you reload the page

As an additional caching argument, you can use unique resource identifiers (ETag). It allows the server to not give the file again, even with ending time caching, and simply extend it. However, there are some problems with the distribution of files on different physical servers and configure them identical ETag, but rather, it refers to the already very large systems. As Alternative the ETag, you can use header Last-Modified.

3.5 Parallel Loading

To reduce the proportion of time for a response from the server when loading a large number of files load can be split into multiple streams (servers) [17]. The servers themselves for that purpose (quick-impact static resources) better customize under the "light" environment (for example, nginx). As a balancing parameter can be considered as the distribution by geography (eg, clusters in the US, Europe, Asia), and load (pool of available servers is determined each time the page is loaded).

Also the use of balancing client to achieve the same effect. The main problems it is worth noting the need to create a hash function of the file name so the same file is loaded with only one server, otherwise the browser will ask the file from a server mirror, until the score cache all copies from all the distributed servers. Also it is necessary to confine 4 hosts (for a large number of files), for a small number (15-25) should be used no more than 3. 2 host used wisely, only if the number of files exceeds 10 because of the additional costs for name recognition in the DNS-table [2].

3.6 Javascript Optimization

It is possible to load required JavaScript-files to display the page, in fact, after loading it. However, there are still a couple of nuances. For example, a web-based resource in this should be fully functional without JavaScript (must be carried out jumps by the links, the initial appearance of the page should be formed on the server). This will increase of indexable the resource by the search engines and will protect those users who have your JavaScript-files do not work for one reason or another (mobile or outdated browsers without JavaScript, and others.). It should rely more on CSS capabilities to create animation effects. You can not display the page in a browser better than the browser itself, so you should leave all the hard work on the drawing page for internal engine (this refers to the animation effects when you hover over the button, layout changes when you resize the window, etc.). CSS-engine, on average, is working faster than JavaScript. Also avoid using CSS-expressions or optimize them so that they performed only once on the page [18].

When optimizing JavaScript-machine interaction with the browser should also update the DOM-tree with large pieces. All DOM-handling resource-intensive, it is a complete analogy to a database server applications. The fewer produced work with DOM, the faster will be performed JavaScript.

It is necessary to make individual comments about the event handlers: their quantity should be kept to a minimum. Ideally, you should use a single on click for the body and already processing source produced action. If you need less than global effects - you can just be limited to one handler on the block, which lies in the desired area. A large number of event handlers (who sometimes forget to remove when changing the containing HTML-code) leads to memory leaks in Internet Explorer.

Also, it may be advisable to cache global variables in the local (however, there may be nuances, especially with the chain of function calls), avoid using eval and setTimeout / setInterval (who perform eval on the argument of a string). Instead, you can use anonymous functions. In the implementation of heavy calculations (for example, additional data loading from the server or sorting large arrays) is worth to update the user interface, so that he knew that performed some action, and could wait. However, the most basic thing here is not to overdo it with notifications: after each update of the page takes some time, and an overhead outgoings because it can be a lot more "useful" time.

3.7 Preloading, Interactive And Full Load

The above techniques will help to greatly accelerate the step of preloading. However, in the presence of a significant number of JavaScript-files, providing animation effects or interaction with users, full load can be significantly delayed in time.

To prevent this, the most common approach is the issuance of an interactive boot (actually, all external JavaScript-files) to preload (with a small amount of code it can be fully included in the source HTML) or post-load (using the combined event window.onload [19]).

As usual, the main criterion will be the audience of web-resource: to optimize page load, it is orienting on its characteristics (average access speed, a typical browser and others.). It should be guided by the following approaches:

1. For constantly updated the audience will be most appropriate to include all files in the source HTML (image - using the combined data: URL approach [20]).
2. If the audience is mixed, it is recommended that about half the size of the page is left in the HTML-file,

and the other half divided into several (4-8) files, which can then be cached.

3. With permanent audience should reduce the size of a HTML-file to a minimum and configure caching headers oriented for a long time.

4. loading all JavaScript files should be made in a fourth step (post-load), thereby speeding up the display of pages in the third stage. Ideally, the second step (interactive load) should not exist: users for some time after the page loads "mastered" on it, get used to the elements of navigation, so at this waiting time, you can load invisibly for users all the interactive elements.

5. For acceleration of loading the necessary background images, you can boost their calls through the dynamic creation of images in the respective head page area (new Image method in JavaScript).

3.8 Non-Blocking Javascript Load

As noted above, the load-JavaScript code on the page should be removed from a number of factors affecting the course of the main (before step 3) loading. Why is this done? JavaScript-code can contain calls document.write (which change the structure of the DOM-tree of the document) or location.href (that redirect to another page). Your browser does not have the right to display the page without having analyzed all the JavaScript-code, so a large number of calls to these files, taking place in the second stage can significantly slow down the loading [21].

If the total size of the JavaScript-code is less than 5% of the total HTML- / CSS-code, it should be included in the latter (located as close to the closing tag body). If JavaScript is a single monolithic block, which provides an interactive operation of the entire page, with its size is large enough for the first case, it is necessary to make this boot code (as a single external file) in the post-loading.

If there are several not related to each other JavaScript-parts they can be called in the post-boot independently (in a few streams through the creation of a dynamic script node in the head-area of the page), which will increase the speed of their load. This applies in particular to the various counters [22].

If the JavaScript on a page is a certain library, which is then used by different applications, and the same library is used on most pages of the web resource, and its using applications vary from page to page, in this case it is necessary to arrange loading by chain. First, in the post-startup is called the library file itself, then it loads all the necessary applications on this page. It refers to the use of most modern JavaScript-frameworks - jQuery, Prototype, Mootools, Ext2, Dojo, YUI.

All of the above methods will help avoid delays in loading related to the use of any JavaScript-code.

IV. CONCLUSION

Summarizing all the above, we can say with firmness: really fast Web resources exist, and create them is not as difficult as it may seem at first glance. The most important thing in the use of client optimization approaches - is to understand at what stage of loading will be affected by this or that action, seeking to maximally speed up the preload page and its basic load.

Algorithms “optimization methods” described above are applicable almost in any situation. It showed extensive practical use for a wide range of tasks: optimization of highly loaded pages [23], the acceleration of the JavaScript-logic on the page [24] and optimization analysis of inhomogeneous Web resources [25].

As practice shows, loading medium Web resource can be accelerated approximately 2-3 times, and all this is achieved by very simple steps: in fact, all that the user does not need right now (right after the preload) can be loaded when displaying the page, or even after the display (within the first 100-500 milliseconds) until the user has not yet committed any active action.

REFERENCES

[1] “Average Web Page Size Triples Since 2003” // Web Site Optimization. Available http://www.websiteoptimization.com/speed/tweak/average-web-page/.

[2] “Best Practices for Speeding Up Your Web Site” // Yahoo!. -Available http://developer.yahoo.com/performance/rules.html.

[3] Bouch, A., Kuchinsky, A., Bhatti, N. , “Quality is in the Eye of the Beholder: Meeting Users' Requirements for Internet Quality of Service” // CHI. - The Hague, The Netherlands : [BN], 2000 r..

[4] “Retail Web Site Performance: Consumer Reaction to a Poor Online Shopping Experience” // Akamai. - 2006 r..

[5] Nah, F., “A study on tolerable waiting time: how long are Web users willing to wait?” // Behaviour. - [BM] : Information Technology, 2004 r.. - 23 : T. 3.

[6] Galletta, D., Henry, R., McCoy, S., Polak, P., “Web Site Delays: How Tolerant are Users?” // Journal of the Association for Information Systems. - 5 : T. 1.

[7] “The Psychology of Web Performance” // Web Site Optimization. - Available http://www.websiteoptimization.com/speed/tweak/psychology-web-performance/.

[8] “CSS Tidy” // Soureforge. – Available: http://csstidy.sourceforge.net/.

[9] “YUI Compressor” // Yahoo!. - Available: http://developer.yahoo.com/compressor/.

[10] «CSS: All about compression» // Web Optimizator. - Available: http://webo.in/articles/habrahabr/14-minifing-css/.
[11] «Javascript: compress or no compress?» // Web Optimizator. - Available: http://webo.in/articles/habrahabr/11-minifing-javascript/.
[12] "Dean Edwards Packer" // Dean Edwards. - Available: http://dean.edwards.name/packer/.
[13] "The influence of the level of gzip compression on server performance" // Web Optimizator. - Available: http://webo.in/articles/habrahabr/33-gzip-level-tool/.
[14] "Replace GIF with PNG Images" // Web Site Optimization. - Available: http://www.websiteoptimization.com/speed/tweak/png/.
[15] « Carefully cut the flow » // Web Optimizator. - Available: http://webo.in/articles/habrahabr/48-flow-slices-optimization/.
[16] «CSS Sprites: all that you know, but were afraid to ask » // Web Optimizator. - Available: http://webo.in/articles/habrahabr/08-all-about-css-sprites/.
[17] "Optimize Parallel Downloads to Minimize Object Overhead" // Web Site Optimization. - Available: http://www.websiteoptimization.com/speed/tweak/parallel/.
[18] « Practical JS: optimizing CSS expressions» // Web Optimizator. - Available: http://webo.in/articles/habrahabr/10-css-expressions-optimization/.
[19] « Practical JS: "postponed" loading » // Web Optimizator. - Available: http://webo.in/articles/habrahabr/05-delayed-loading/.
[20] « Cross-browser usage data:URL» // Web Optimizator. - Available: http://webo.in/articles/habrahabr/46-cross-browser-data-url/.
[21] "Non-blocking JavaScript Downloads" // Yahoo! User Interfacses Blog. - Available: http://yuiblog.com/blog/2008/07/22/non-blocking-scripts/.
[22] « We accelerate counters from myth to reality » // Web Optimizator. - Available: http://webo.in/articles/habrahabr/61-counters-optimization/.
[23] « Optimize pages often show » // Web Optimizator. - Available: http://webo.in/articles/clientside2007/common-pages-optimization/.
[24] "High Performance Ajax Applications" // Julien Lecompte Blog. Available: http://www.julienlecomte.net/blog/2007/12/39/.
[25] « Optimization Results » // Web Optimizator. - Available: http://webo.in/about/results/.