Large Scale Web Page Optimization of Virtual Labs

***Abstract*—We propose web page optimization as necessary step in web application development. In context of virtual labs, several online experiments were designed by domain experts from various disciplines and thousands of web pages were developed. But domain expert did not know best practices for web development, therefore web pages had lot of performance issues on client side. So end user suffered slow rendering of pages on the client side. Performance of an web application largely depends upon the content of the web page because rendering takes place on client side. And client machine may have limited resources at his side. Therefore web page optimization seems necessary step to improve user experience. In contrast to existing work on this field, here we focus on large scale web performance analysis through data visualization. Based on this large scale visualization, we present a utility of existing web optimization tool by experimentation on a MHRD web project "virtual labs".**

I. Introduction

After the onset of the world wide web, web performance implied optimizing the server side but nowadays optimization on the client side has become bottle neck because of complex web application. Now days, front-end developers use a lot of critical resources like javascipt, CSS and images to make a good user interface which becomes overhead during page rendering which means lesser user experience. Success of a web application is judged based on user's experience which also includes fast response time. Optimized web pages not only render a web page fast but also saves network bandwidth. Along with a good responsive web page, web developers should also focus on optimizing critical resources by minifying size and number of resources Critical rendering path is the chain of necessary events that occurs to render webpage on browser. The main critical resources on a web pages are CSS, javascript and images. For each critical resource on a webpage, browser makes a new request to server. CSS and javascript are the two critical resources that blocks the rendering of a webpage. Therefore, correct sequence of http requests can reduce the perceived page load time which means page load time will not reduce in reality but critical resources needed to render webpage are fetched first and remaining resources are fetched in background. Therefore, while optimizing web page for speed we should focus on [10]

1) Minimize the size of critical resources
2) Minimize the number of critical resources
3) Minimize the critical rendering path.

The goals of our experimentation, performed in collaboration with VLEAD, IIIT-Hyderabad were :

- To analyze performance of thousands of web pages hosted at virtual labs.
- To evaluate the utility of the Pagespeed with respect to optimizing web pages of virtual labs.
- Comparison between the web performance of virtual labs web pages with and without a web optimization like Google PageSpeed tool.

II. Background

Initially when world wide web started most of the web pages where plain hyper text documents. Slowly people started sharing media content like images and audio on web and in past decade web pages got transformed into web application with depends upon on critical resources like CSS, javascript and media content like images, audio, video etc. Now rendering of web pages depends upon following three major criteria‘s. Firstly order in which critical resources are fetched, secondly on number of critical sources and lastly on size of critical resources. Web gaints like Yahoo and Google have listed best practices for web developers to improve performance of web pages. So now we briefly explain some of the best practices in order to appreciate significance of web performance optimization [6].

- **Minimize http requests :** Number of http requests to render any web page is directly proportional to number of critical resources. For each critical resource in the page, browser have to make a new http request to server and then its get loaded. So, almost 80% of the response time is consumed in downloading all the resources. So to reduce number of http requests, one can combine multiple CSS into one, also multiple javascipt files can be combined into one. Other ways include image spriting, etc.
- **Use a Content Delievery Networks :** The nearest geographical server is selected for delivering the content which reduces load time.
- **Add Cache Control Header :** For static components, setting far future expires header will reduce number of http request so that browser can fetch resources from cache when request to same webpage is made again. For dynamic components, use an appropriate Cache control header to help browser with conditional requests. This reduces unnecessary http requests.
- **Gzip components :** Compression reduces response time by reducing the size of http response. Gzip is most popular and effective compression method at this time. It reduces response time by 70%. If a web client indicates for support for compression in the http request header, server sends a compressed components.
- **Stylesheets at the Top:** Problem with not including link to style sheets in the head tag is that it blocks progressive rendering and till the style sheets are not fetched users sees nothing on screen.

- **Put Scripts at Bottom :** Putting scripts at the top, blocks parallel downloading of resources from same host. In most of the cases scripts are needed when users starts interacting with the web page.
- **Make Inline Small CSS and javascript files :** If file size is to small, then it should be made inline as it will reduce number of http requests.
- **Make large css and javascript file external :** CSS and javscript files are cached by browser. So we avoid downloading it every time a request is made. First time it will take time to download but all later requests are served from the user's browser cache.
- **Minify javascript and css :** Unnecessary characters from code should be reduced which includes removing comments, redundant code and removing white spaces. Removing this unnecessary characters reduces number of critical bytes served to client-side and increases response time.
- **Avoid Redirects :** Connecting an old page to new one takes leads to an extra cycle of dns lookup and tcp handshake adding latency and delaying the response. Therfore, it should be avoided.
- **Configure ETags :** Entity tags is a way that browser and server use to determine whether the component in cache is same as that on server.
- **Flush the buffer early :** It allows to send partially ready response to browser.It should be written as early as possible in the code, preferably in the head section. In php there is function flush() to flush the buffer.
- **No 404 error :** Http requests are expensive and getting a response like '404 Not found' is totally useless and it increases latency time. Therefore, all such web pages should be avoided.
- **Make favicon small and cacheable :** Favicon stays in the core of server and it is necessary because if its missing, browser will still request for it and adding to the latency.

Yahoo **Yslow** [4] and **Google Pagespeed** [11] are well known tools that are capable of evaluating web pages performance and providing suggestions to optimize web pages based on the best practices listed above. But the pagespeed is more than a performance analysis tool and is also capable of optimizing web pages without changing semantics of a webpage.

Yahoo yslow.js is a javascript API which runs on phantomjs. PhantomJS is a headless browser with JavaScript API [7]. We used Yslow as performance measuring tool because it not only analyzes a webpage but gives suggestions on how to improve it. It works on following three ways:

- It crawls the DOM to find each component.
- Collects information for each component and analyzes each of them.
- It generates scores out of 100 for each rule which produces the overall score for page.

The grades for individual rules are computed differently depending on the rule. For example, for Rule 1, three external scripts are allowed and each extra script four points are deducted from the overall weight. The code for grading each rule is found in rules.js. The overall grade is a weighted average of the individual grades for each rule, calculated in controller.js The rules are approximately in order of importance, most important first. The specific weights are in the ruleset objects in rules.js. For score computation please refer [8].

Mod_pagespeed [12] is an automated web page optimization tool developed by Google for optimization of web pages. It not only analyzes a web page but also optimizes it. Based on best practices, it has certain set of filters which optimizes web page during run time. As the server gets request for the webpage, it dynamically rewrites the page using its filters [13] and sends a highly optimized page. The rewriting happens only when the first request for web page is made and for all later request web page is served from the cache at server side. There are total of 40+ filters which supports optimization. These filters can be turned on or off based on requirements.

Score	Grade
95-100	A+
90-94	A-
85-89	B+
80-84	B-
75-79	C+
70-74	C-
65-69	D+
60-64	D-
55-59	E+
50-54	E-
Below 50	F

III. Approach

Our work is broadly divided into four major phases namely Data Collection, Data Visualization, Analysis of Data and optimizing web pages based on analysis. During data collection phase we first collected all the urls of virtual labs hosted at IIIT-Hyderabad. Then we collected Yslow reports for each web page using an automated script and phantomJS. After collecting all the reports we extracted scores for each rule from reports and stored it in a csv file. During visualization phase all data is visualized using an automated script indicating performance for each rule and also overall performance of web pages. Later we did analysis for optimized pages of virtuals labs.

A. Urls and Report Collection

As first step to collect data and generate reports, we collected urls of all the web pages at following domain deploy.virtuals-labs.ac.in hosted at IIIT-Hyderabed. Since we have access to server, to get all the urls, we wrote an automated bash script to extract all the html and php pages links from the server and stored it into the text file. To test the performance of vlab.co.in, we also collected the 5000 urls for this website. This time since we do not have the access of server ,we used an online sitemap generator to get the urls. We wrote an automated script to generate the reports for each web page using yslow.js and phantomJS. This automated script read a url from a file containing all the urls and generate report for each web page. In this script, we are making ten phantomjs call as background process for ten different web pages. For urls which are inactive, failed reports will be generated and we are pushing that url into the failed urls file and deleting the failed reports. These reports serves as input to CSV file generation.

B. CSV File Generation

CSV file are generated using a bash script. This CSV file will contain the overall score and scores corresponding to each rule. Script will extract all the scores for corresponding rule and will dump into the CSV file line by line. The content in the CSV file will be used for visualization.

C. Visualization

Visualization is done using python matplotlib and csvkit. We have written an automated scripts to generate all the graphs needed for visualization. Scripts takes csv file as input and set of bar graphs specific to each rule.

IV. Experimentation

A. www.vlab.co.in

We started our experiment by collecting Yslow reports for vlabs. We collected 5000 urls of www.vlab.co.in. and generated Yslow reports corresponding web pages and generated CSV file for the scores. Finally, we plotted data accumulated in csv files and analyzed the current status of web pages by looking at bar graphs.

B. deploy.virtual-labs.ac.in without pagespeed

This is the virtual lab website hosted at IIIT-Hyderabad. To analyze performance of web pages, we collected all the urls located at deploy.virtual-labs.ac.in by using our bash script. Total of 8786 urls were collected, reports were generated for each web page located at respective url and scores were accumulated in a csv file. Later we visualized the scores to analyze the current status of web pages.

C. Optimization of deploy.virtual-labs.ac.in using mod_pagespeed

We made a replica of deploy.virtual-labs.ac.in server on a container. We installed Google mod_pagespeed on the container to see what extent pagespeed optimizes. Then, we collected reports for all the web pages and generated CSV file. After generating bar graphs we compared results with deploy.virtual-labs.ac.in without pagespeed to observe how much optimization was done by pagespeed.

V. Analysis

In this section we analyze web page performance of virtual-labs.ac.in in two different settings on data set containing 9000 web pages. One when page speed module is installed on our system and other one without page speed. We also analyze web page performance of vlab.co.in on data set containing 5000 web pages.

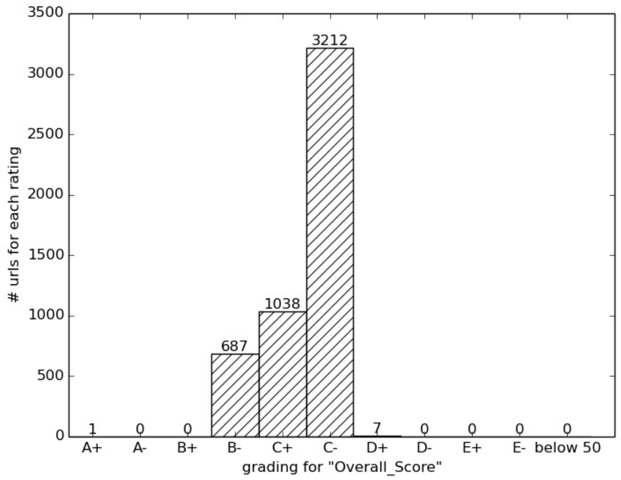


Fig. 1: Number of urls versus Overscore

A. Analysis of vlabs.co.in

Out of 5000 urls we collected statistics for 4945 web pages.

- From the bar graph in figure 1, we can observe that only 1 web page is having good performance. Rest of the web pages does not get good grades and need to be optimized to improve performacne. Out of rest 687 i.e 13% web pages are in above average conditions, 1038 web pages are performing average and rest 3220 web pages are not in good condition. Therefore, these statistics show that these pages are not following performance best practices and performance of web pages can be improved.
- From the bar graph in figure 9, we can observe that all the web pages are having their CSS files inside the head tag else they block the progressive rendering.
- From the bar graph in figure 10, we can observe that all the web pages are having their script files at the bottom else they block parallel downloading.
- From the bar graph in figure 11, we can observe there is no usage of Content Delivery Networks(CDN). It suggested to use CDN's for large web sites like www.vlab.co.in in order to reduce latency.
- From the bar graph in figure 12, we can observe that there is no use of Expires Headers in the web pages. If this practice is not followed in the pages and it has to fetch each static critical resources every time page is requested.
- From the bar graph in figure 13, we can observe that only 89 web pages out of 4945 pages are in compressed form and rest are in uncompressed form. This will lead to large payload size and will increase network traffic and will slow page rendering.
- From the bar graph in figure 14, we can observe that this website does not use entity tags. Due to this it

has to download critical resources every time request for page is made.

- From the graph in figure 15, we can see that all pages vlabs are having favicon for it.
- from the graph in figure 16, we can see that number of critical resources on the pages are more than required. Only 1 web page is getting A+ grade. Around 4500 web pages have scores between 65 to 75 means they are loaded with multiple css, javascript and images. For each resource, browser have to make a request to server and most of the time is wasted downloading the resources. This resources number should be reduced for fast rendering.

B. Optmization using Pagespeed.

As we have generated statistics for 8986 web pages under deploy.virtual-labs.ac.in without pagespeed and for replica of deploy server with pagespeed installed. Here we observed how much pagespeed optimizes the web pages in comparison when pagespeed is not used. Note that all figures are palced in the Appendix Section.

- From the bar graph in figure 2 we can notice that 4299 web pages fall under A+ grade for **overall score** while with pagespeed this rises to 6051. It shows pagespeed is improving the performance of web pages by optimizing it. Also we can see see without pagespeed only 1226 wepages are in A- grade but with pagespeed it increased to 1667. From the figure 2 we can observe that overall performance of web pages has improved after using pagespeed.
- From the bar graphs in figure 3, we can observe that without pagespeed only 3214 web pages are in B+ use **added expires headers** and around 45% are not using it. But with pagespeed, 4729 web pages were in B+ grade and there was shift of number from low grades to high grades.
- From the bar graphs in figure 4, we can observe that most web pages follows **compressed using Gzip** rule and 8257 urls falls under A+ grades while with pagespeed this number rises to 8761.
- From the bar graphs in figure 5, we can observe that **Etags rule** without pagespeed 3535 pages have A+ grade, 831 have B+ grade, 384 have C+ grade, 3704 have score below than 50 but with pagespeed this number shifted to more higher grades. We can see now 6441 pages fall in A+ compared with 3534 pages without pagespeed. But the major difference is seen at below 50 score pages, now this have reduced to 633 in comparison to previous 3704.
- From the bar graphs in figure 7, we can see that w.r.t **make fewer http requests**, without pagespeed only 7791 web pages were in grade A+ and 461 pages were in A- but with pagespeed 8202 pages were in A+ grades. Major shift was from A- grade to A+ grade. Rest pages shows no observable changes.
- From the bar graphs in figure 8, we can see that **minified JavaScript and CSS** rule without pagespeed 7059 web pages were in grade A+, 1395 pages in A-, then 254 in B- and 20 were below F. After optimizing with pagespeed 8639 pages were in grade A+ ,94 in A- and only 3 below F.

VI. Conclusion

This paper is concerned about Web Page Optimization of virtual labs which means fast rendering and less network bandwidth. This lead us to think us how to decrease the number and size of resource and also how to decrease the perceived load time. These web pages consists of critical resources like CSS, JavaScript and images. This optimization can be achieved by minimizing the number of critical resources, minimizing the size of critical resources and minimizing the critical path length.

Our framework is helps in visualizing website performance and it can be used to suggest lab developers to work on these best practices. Also it gives the list of urls pages which are broken. Google Pagespeed is the automated web optimization tool that comes as a saviour for optimizing web pages of already existing web applications. It has 40+ filters which optimizes page and can be used according to our requirement. Also, it is open source and available for free. It should be used for huge websites like virtual labs application developers comes various domains and backgrounds.

VII. Future Work

This framework can be modified to give the list of urls of web pages that perform badly. Generating report takes at least 24 hrs to process 5000 urls but by making phantomjs clusters on framework like Hadoop we can parallelized the whole report generation. Also functionality of pagespeed can be improved by adding more filters. For example, there is no filter to give default favicon for a webpage.

References

[1] Andrew B.King, 2008. Website Optimization : Web performance Optimization. 155-185,282-290.

[2] Steve Souders, 2007. High Performance Websites. 10-84

[3] Steve Souders, 2009. Even Faster Web Sites

[4] Yslow Official Website.Available:
http://yslow.org/phantomjs/

[5] Yslow Documentation Page:
http://yslow.org/faq/

[6] Yahoo Developer Network. Best Practices for Speeding Up Your Website:
https://developer.yahoo.com/performance/rules.html

[7] PhantomJs:
http://phantomjs.org/

[8] Yslow Score Computation:
http://yslow.org/ruleset-matrix/

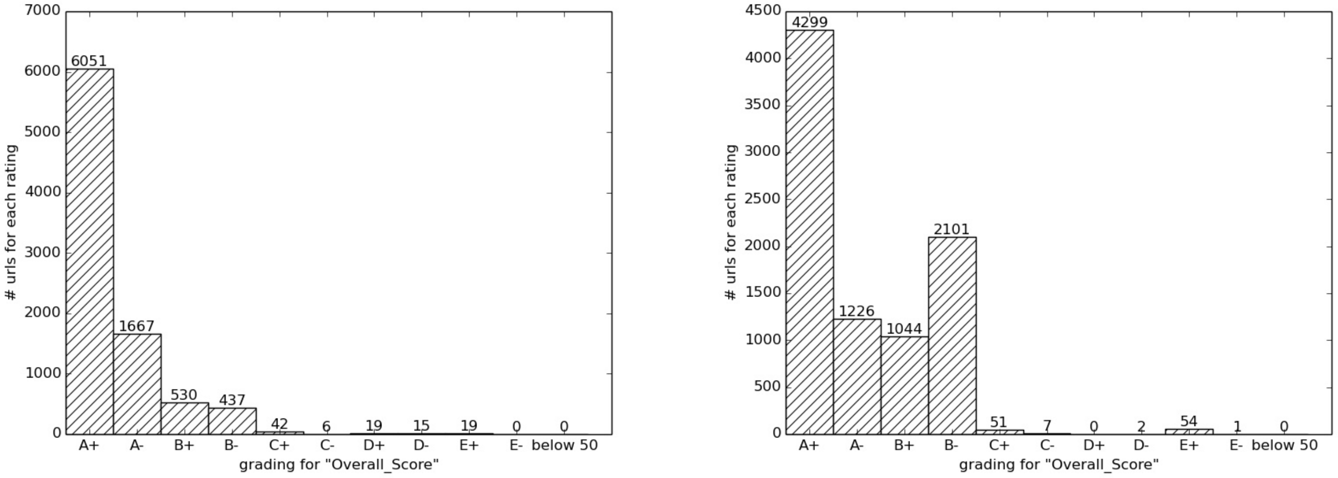
[9] Google Developers. Analyzing The Critical Rendering Path
https://developers.google.com/web/fundamentals/performance/critical-rendering-path/analyzing-crp

[10] Google Developers. Optimizing The Critical Rendering Path:
https://developers.google.com/web/fundamentals/performance/critical-rendering-path/optimizing-critical-rendering-path

[11] Google Pagespeed Tools.Official Documentation:
https://developers.google.com/speed/pagespeed/

[12] Google Mod_pagespeed.Available:
https://developers.google.com/speed/pagespeed/module

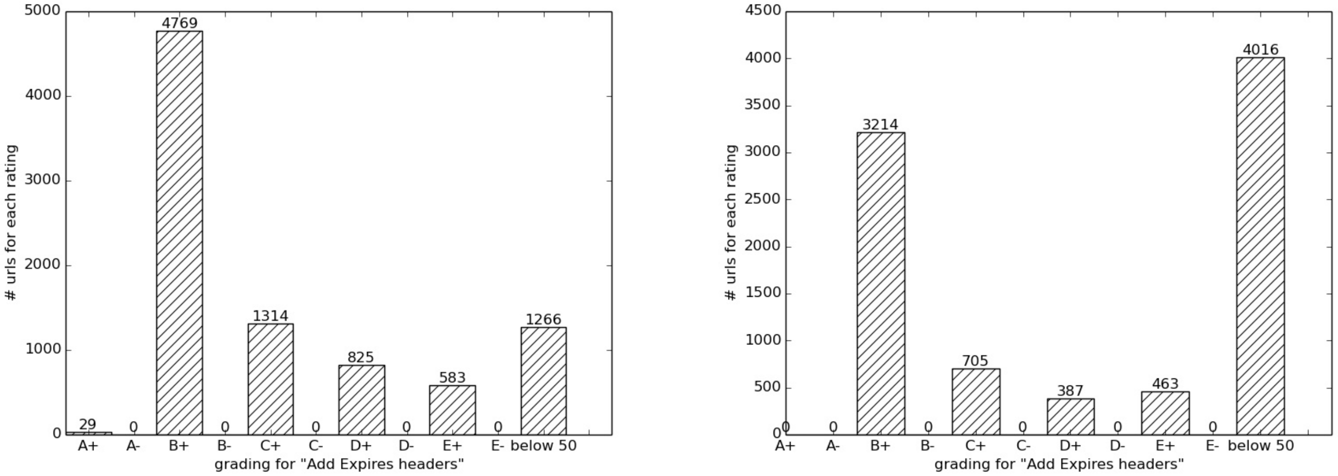
[13] Pagespeed Filters:
https://developers.google.com/speed/pagespeed/module/filters



(a) Overall Score With PageSpeed

(b) Overall Score without PageSpeed

Fig. 2: Comparing Overall Scores

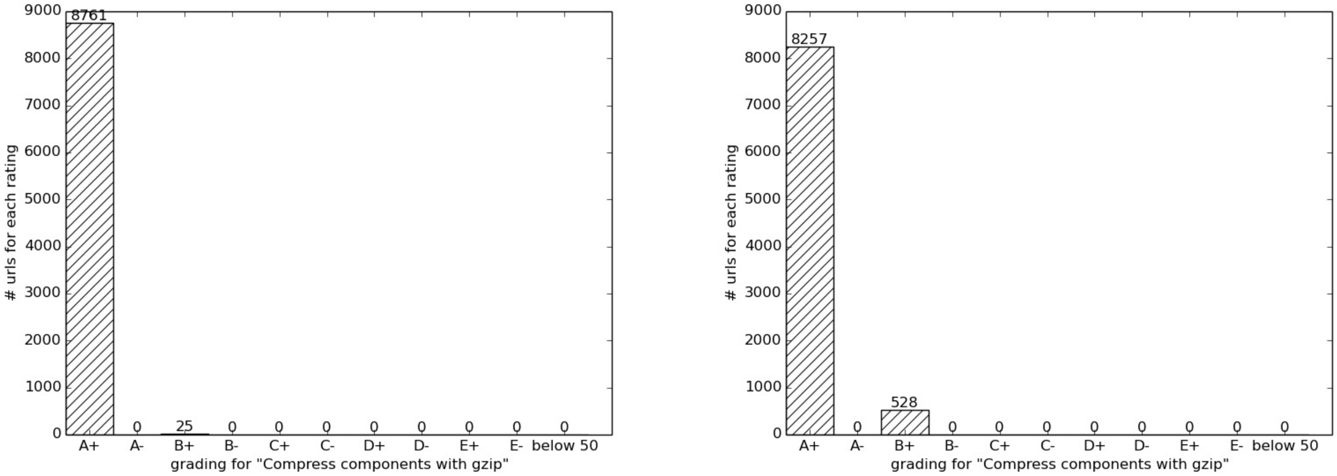


(a) 'Add Expires Header' Rule with PageSpeed

(b) 'Add Expires Header' Rule without PageSpeed

Fig. 3: Comparing 'Add Expire Header' rule

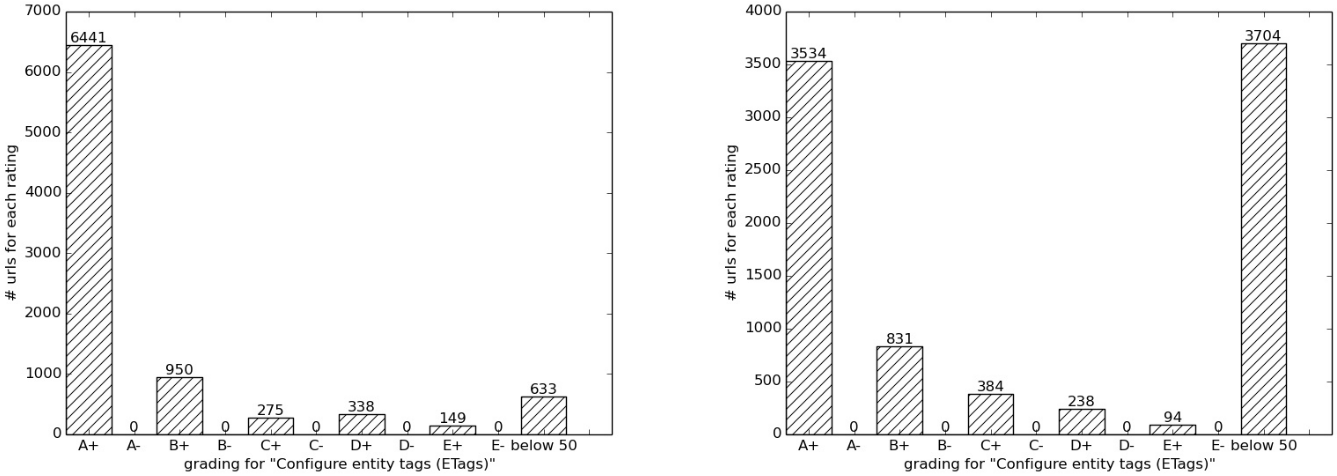
APPENDIX



(a) ’Compress Components with gzip’ rule with PageSpeed

(b) ’Compress Components with gzip’ rule without PageSpeed

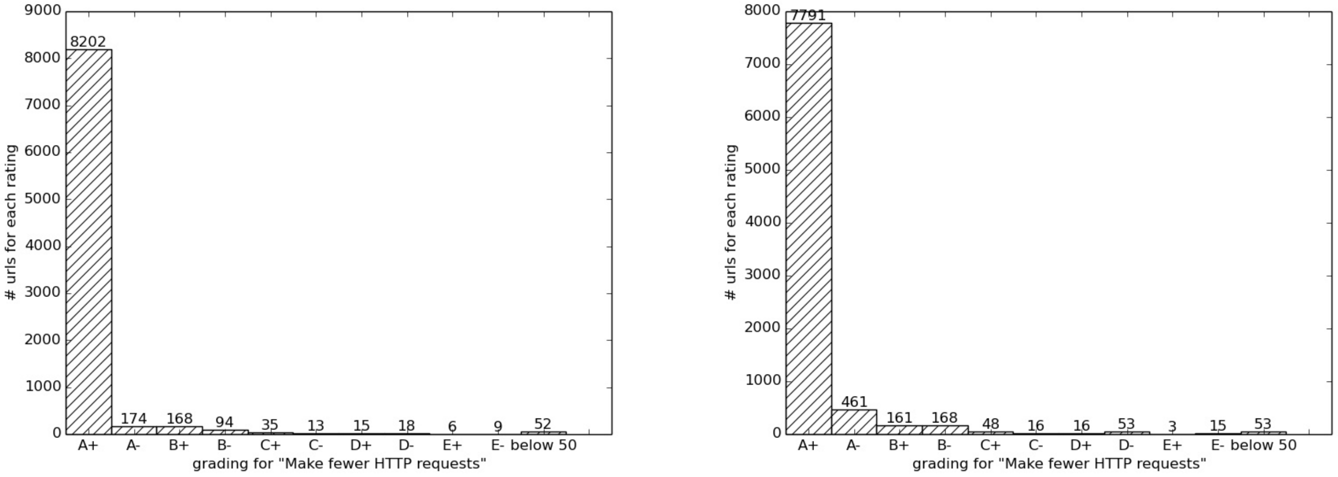
Fig. 4: Comparing ’Compress Components with gzip’ rule



(a) ’Configure entity tags (ETags)’ rule with PageSpeed

(b) ’Configure entity tags (ETags)’ rule without PageSpeed

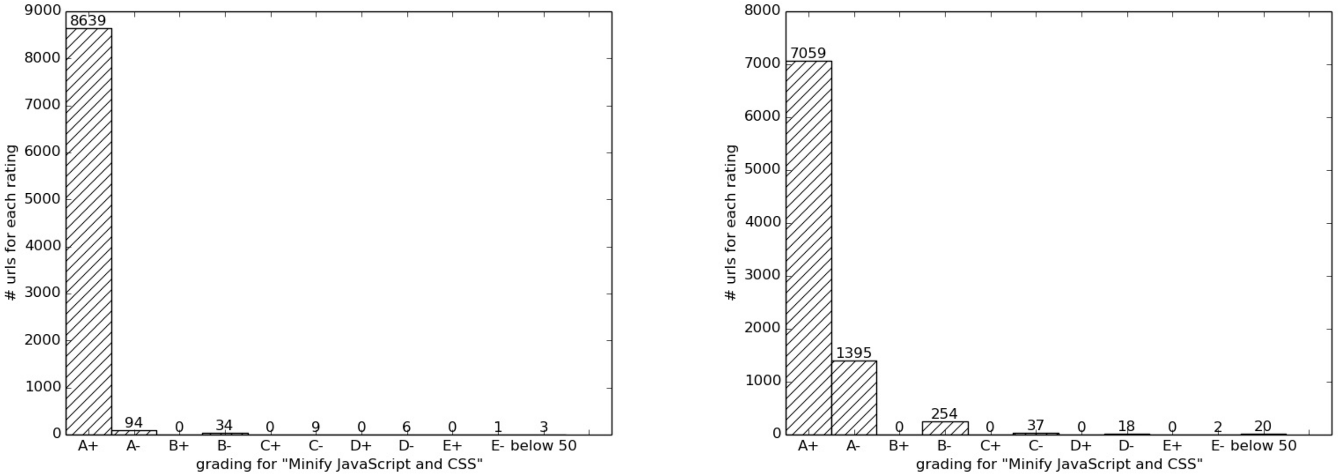
Fig. 5: Comparing ’Configure entity tags (ETags)’ rule



(a) 'Make fewer HTTP requests' rule with PageSpeed

(b) 'Make fewer HTTP requests' rule with PageSpeed

Fig. 7: Comparing "Make fewer HTTP requests' rule



(a) 'Minify JavaScript and CSS' rule with PageSpeed

(b) 'Minify JavaScript and CSS' rule without PageSpeed

Fig. 8: Comparing 'Minify JavaScript and CSS' rule

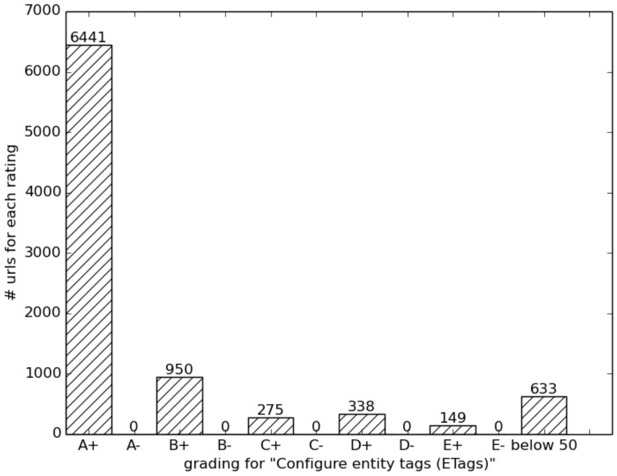


Fig. 6: 'Number of urls' versus 'Configure entity tags (ETags) on container'

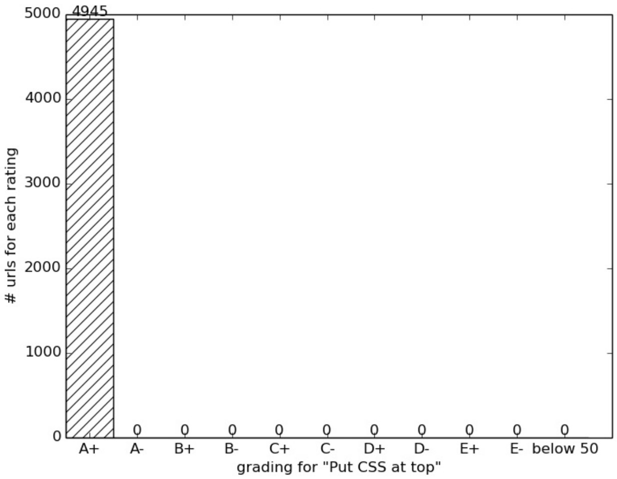


Fig. 9: Number of urls versus Put CSS at top

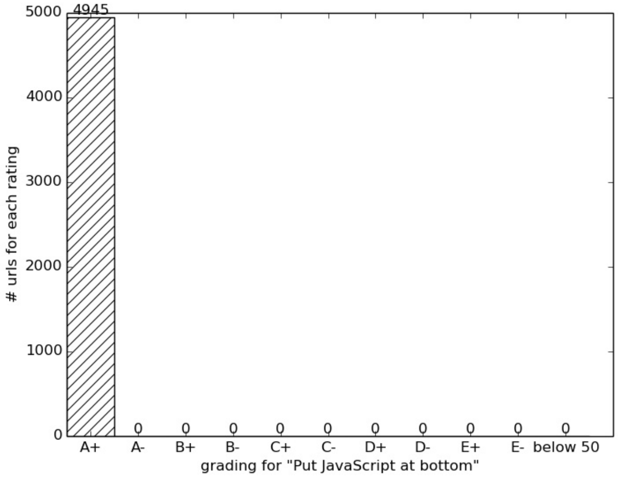


Fig. 10: Number of urls versus Put JavaScript at bottom

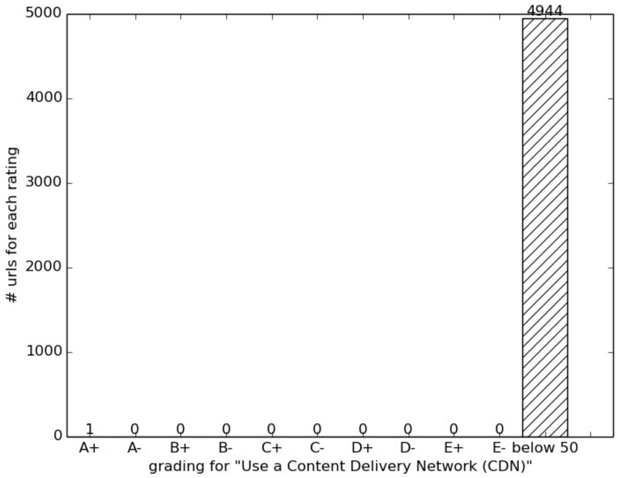


Fig. 11: Number of urls versus Use a Content Delivery Network (CDN)

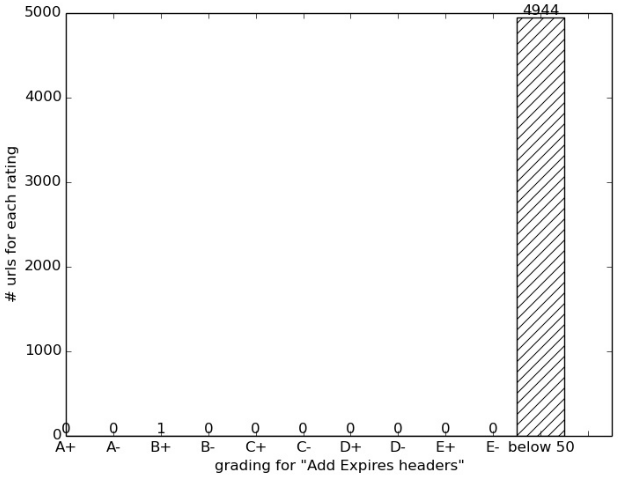


Fig. 12: Number of urls versus Add Expires Headers

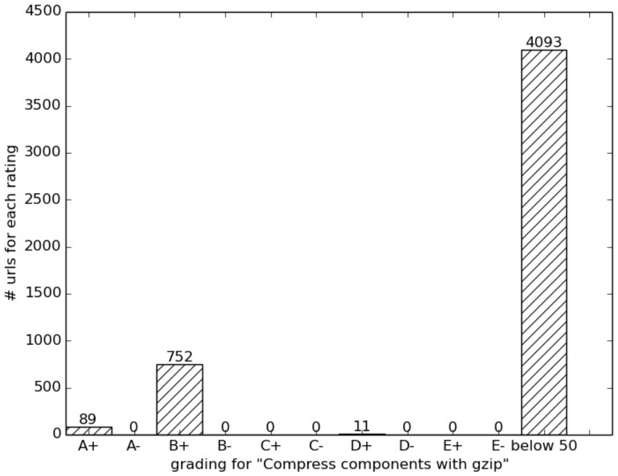


Fig. 13: Number of urls versus Compress components with gzip

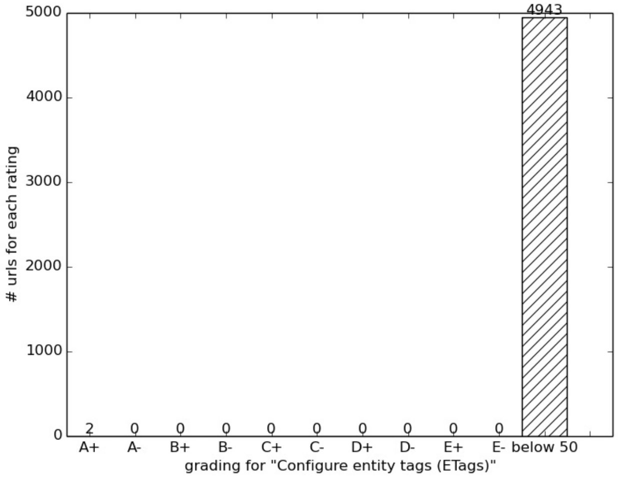


Fig. 14: Number of urls versus Configure entity tags (ETags)

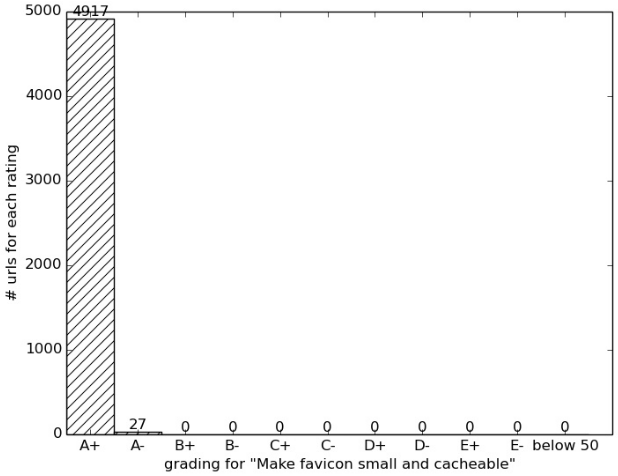


Fig. 15: Number of urls versus Make favicon small and cacheable

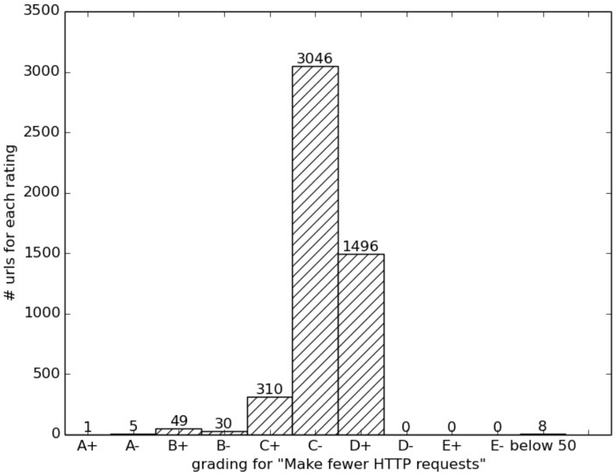


Fig. 16: Make fewer HTTP requests Versus Number of urls