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 web applications were designed by people who are experts in their respective disciplines and hundreds of virtual labs were developed as a result. Based on complexity of each virtual lab, each lab developer decided to use his own set of tools and technologies, therefore virtual lab had lot of performance issues. So end user suffered from slow rendering of pages on the client side. Performance of a 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 on his side. Therefore web page optimization seems a necessary step to improve user experience. In contrast to existing work in this field, here we focus on large scale web performance analysis through data visualization. This visualization gives a us an understanding of performance issues of virtual labs. Based on this large scale visualization, we present a utility of existing web optimization tools by experimentation of an MHRD web project "virtual labs".**

I. Introduction

After the onset of the world wide web, web performance implied optimizing the server side but according to current trends optimization on the client side has become bottle neck because of complex web applications. *Critical rendering path* is the chain of necessary events that occur to render web page on browser and resources fetched during critical rendering path are called *Critical Resources* [1]. Front-end developers use too many critical resources like JavaScipt files, CSS files and images to make a good user interface which becomes overhead during page rendering which means indigent user experience. Success of a web application is judged based on user's experience which also depends upon fast response time. Optimized web pages not only render fast but also save network bandwidth. Along with a good responsive web page, web developers should also focus on optimizing critical resources by minimizing size and number of resources. For each critical resource on a web page, browser makes a new request to server. CSS and javascript are the two critical resources that block the rendering of a web page. Therefore, correct sequence of http requests can reduce the perceived page load time which means page load time does not reduce in reality. So critical resources needed to render web page should be fetched first and remaining resources should be fetched in the background. Therefore, while optimizing web page for speed we should focus on [2]

1) Minimizing the size of critical resources
2) Minimizing the number of critical resources
3) Minimizing 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 *Pagespeed* in order to optimize web pages of virtual labs.
- Comparison between the web performance of web pages of virtual labs with and without a web optimization tool like *Google PageSpeed*.

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 which depend 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 the significance of web performance optimization, for details explanation please refer to Yahoo's Best practices [3].

- **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 has to make a new http request to server and then it gets 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 javascript files can be combined into one. Other ways include image spriting, etc.
- **Use Content Delievery Networks :** The nearest geographical server is selected for delivering the content which reduces network latency.
- **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 web page 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 support for compression in the *http request header* then 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 start interacting with the web page and not during the rendering 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 the 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 takes time to download but all later requests are served from the *browser cache*.
- **Minify javascript and css :** Unnecessary characters from code should be reduced which includes removing comments, redundant code and removing white spaces. Removing these unnecessary characters reduces total number of critical bytes served to client-side and increases response time.
- **Avoid Redirects :** Connecting an old page to new one 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 a 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 adds to latency. 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 then browser will still request for it and that adds to latency.

Yahoo **Yslow** [4] and **Google Pagespeed** [5] 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 web page.

Yahoo yslow.js is a javascript API which runs on phantomjs. PhantomJS is a headless browser with JavaScript API [6]. We used Yslow as performance measuring tool because it not only analyzes a web page but gives suggestions on how to improve it. It works on following three ways:

- It crawls the DOM to find each component.
- Collects information from 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 applied approximately in order of importance, most important first. The specific weights are in the ruleset objects in rules.js. For details on score computation please refer to Rule Matrix [7].

Mod_pagespeed [8] 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 web page, it dynamically rewrites the page using its filters [9] and sends a highly optimized page. The rewriting happens only when the first request for web page is made and for all later requests web page is served from the cache at server side. There are more than 40 filters which support optimization. These filters can be turned on or off based on requirements.

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 on www.deploy.virtual-labs.ac.in. 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 carried out analysis to optimize the web pages of virtual labs.

A. Urls and Report Collection

As a first step to collect data and generate reports, we collected urls of all the web pages at following domain deploy.virtuals-labs.ac.in. Since we have access to server, to get all the urls, we extracted all the html and php pages' links from the server and stored it into the text file using an automated bash script. In order to test the performance of web pages www.vlab.co.in, we collected the 5000 urls using an online sitemap generator since we did not had access to server. We extracted *yslow* report for each web page using yslow.js and phantomJS using automated bash script. This automated script read the url line by line from a text file containing urls and generated a report for each web page. This script does batch processing running ten phantomjs processes in background for ten different web pages. The urls of dangling web pages are pushed into a text file named failed-urls and such reports are deleted. These *ylsow* reports serves as input to CSV file generation.

B. CSV File Generation

CSV file are generated using a bash script. This CSV file contains the overall score and scores corresponding to each rule. Script extracts all the scores corresponding to each rule and dumps it into the CSV file line by line. The content in the CSV file is used for visualizing statistics.

C. Data Visualization

Data visualization is carried out using python matplotlib [10] and csvkit [11]. Statistics for each rule is plotted using an automated script in the form of bar graphs as shown in Appendix A. Scripts take csv files as input and return set of bar graphs specific to each rule as output.

IV. Experimentation

In order to achieve our goals we carried out experimentation on three different set of urls:

1) set containing 4945 urls of *www.vlab.co.in* without *pagespeed*
2) set containing 8786 urls of *www.deploy.virtual-labs.ac.in* without *pagespeed*
3) set containing 8786 urls of *www.deploy.virtual-labs.ac.in* with pagespeed

For each of the above set of urls we generate *yslow* reports and accumulate scores in three different CSV files. Later statistics are visualized in the form of bar graphs for each csv file. We analyze the performance of www.vlab.co.in using obtained bar graphs in next section. We also studied comparison between performance of 8786 web pages of www.deploy.virtual-labs.ac.in with and without *pagespeed*.

V. Analysis

In this section we analyze web page performance of *www.virtual-labs.ac.in* in two different settings on a data set containing 8786 web pages. One when page speed module is installed on our system, other one without it. We also analyze web page performance of vlab.co.in on data set containing 4945 web pages.

A. Analysis of performance for 5000 web pages of www.vlab.co.in

Out of 5000 urls, we collected statistics for 4945 web pages. Note that all figures are placed in the Appendix Section.

- From the bar graph in figure 1, we can observe that only 1 web page is having good performance. Rest of the web pages do not get good grades and need to be optimized to improve performance. Out of rest 687 i.e 13% of the 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 best practices and performance of web pages can be improved.
- From the bar graph in figure 2, we can observe that all the web pages are having their CSS files inside the head tag else they would block the progressive rendering.
- From the bar graph in figure 3, we can observe that all the web pages are having their script files at the bottom else they would block the parallel downloading.
- From the bar graph in figure 4, we can observe there is no usage of Content Delivery Networks(CDN). It is 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 5, we can observe that there is no use of *Expires Headers* in the web pages. If this practice is not followed in the web pages then it has to fetch each static critical resource every time page is requested.
- From the bar graph in figure 6, 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 7, 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 8, we can see that all web pages of vlabs are having favicon for it.
- From the graph in figure 9, 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 meaning that they are loaded with multiple css, javascript and images. For each resource, browser has to make a request to server and most of the time it is wasted downloading the resources. Number of resources should be reduced for fast rendering.

B. Comparison with and without Pagespeed

We have generated statistics for 8786 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 placed in the Appendix Section.

- From the bar graph in figure 10 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 about 20%. Also we can see that without pagespeed only 1226 web pages are in A- grade but with pagespeed it increased to 1667. From the figure 10 we can observe that overall performance of web pages has improved after using pagespeed.
- From the bar graphs in figure 11, 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 12, we can observe that most web pages follow **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 13, 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 14, we can see that w.r.t **make fewer http requests** rule, 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 15, 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 and Future Work

This paper is concerned about Web Page Optimization of virtual labs in order to improve user experience. This optimization is achieved by minimizing the number of critical resources, minimizing the size of critical resources and minimizing the critical path length. Our framework helped us in visualizing web performance and provide suggestions to developers to improve performance. It also gives the list of urls of dangling web pages. Our comparison study asserts that Google Pagespeed as strong utility for optimizing web applications. It should be used for huge websites like virtual labs, where various domains and backgrounds are involved.

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 parallelize the whole report generation. Also, the functionality of pagespeed can be improved by developing more filters. For example, there is no filter to give default favicon for a web page.

References

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

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

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

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

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

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

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

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

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

[10] Matplotlib: http://matplotlib.org/

[11] CSV Kit: https://csvkit.readthedocs.org/en/0.8.0/

Appendix

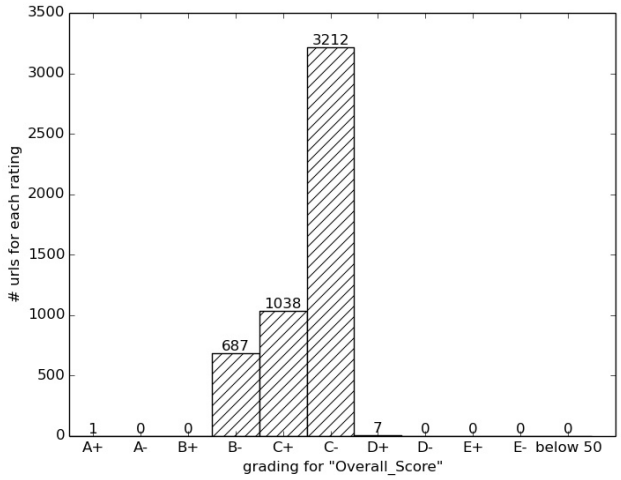


Fig. 1: Number of urls versus Overscore

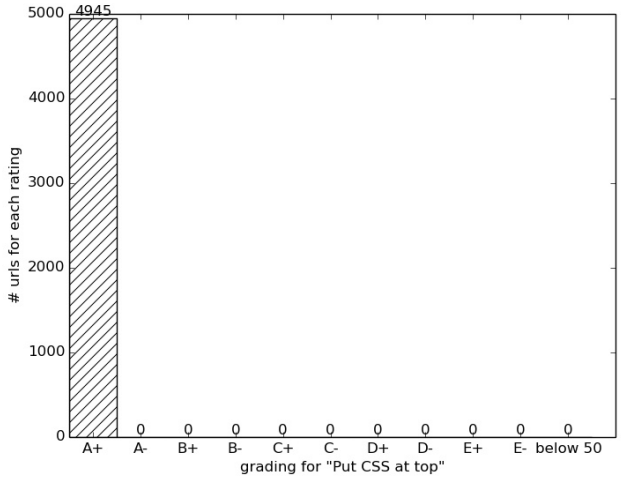
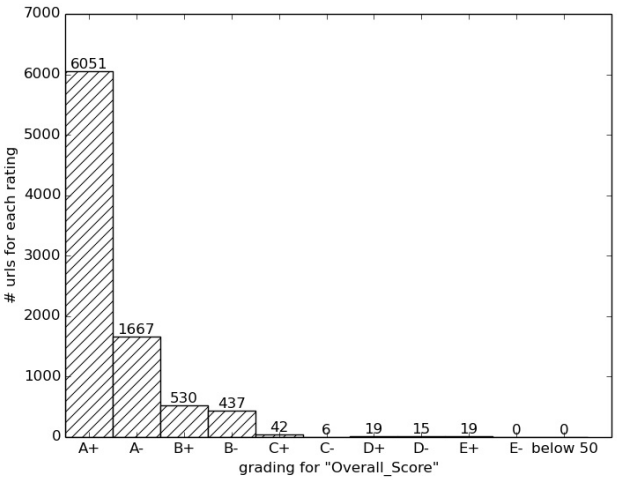
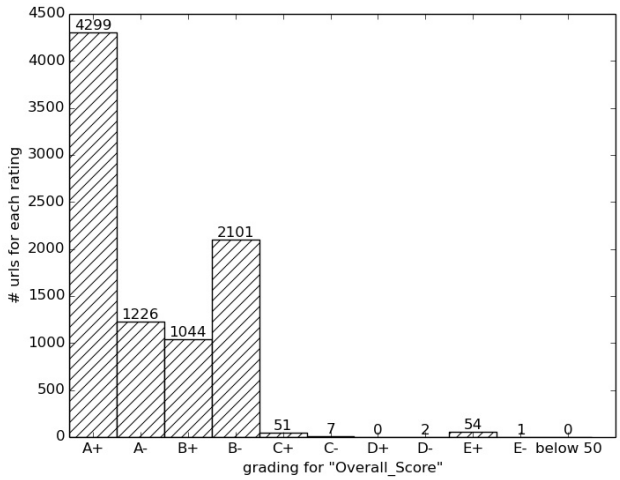


Fig. 2: Number of urls versus Put CSS at top

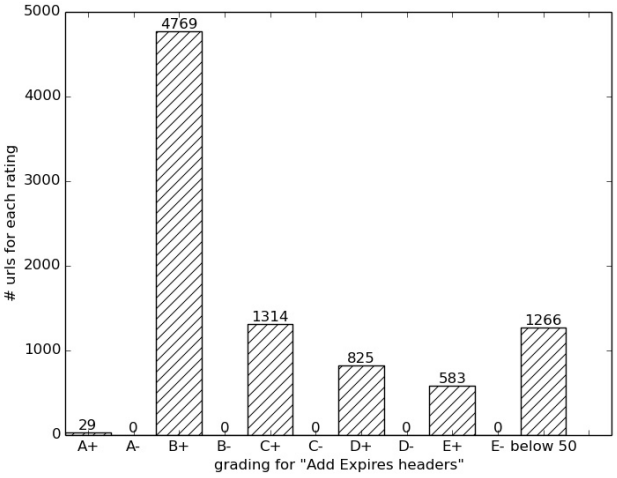


(a) Overall Score With PageSpeed

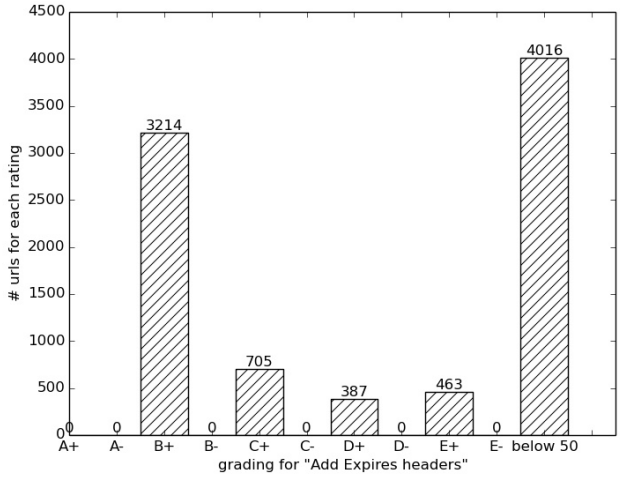


(b) Overall Score without PageSpeed

Fig. 10: Comparing Overall Scores

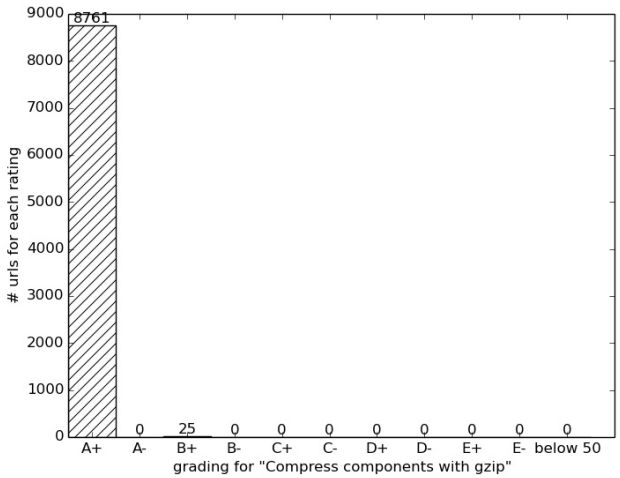


(a) 'Add Expires Header' Rule with PageSpeed

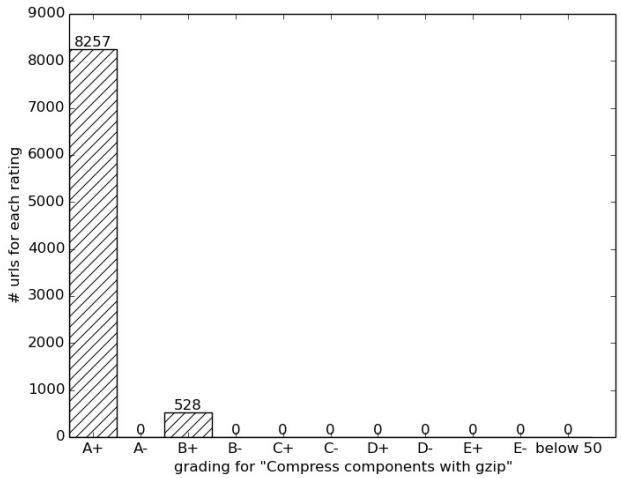


(b) 'Add Expires Header' Rule without PageSpeed

Fig. 11: Comparing 'Add Expire Header' rule

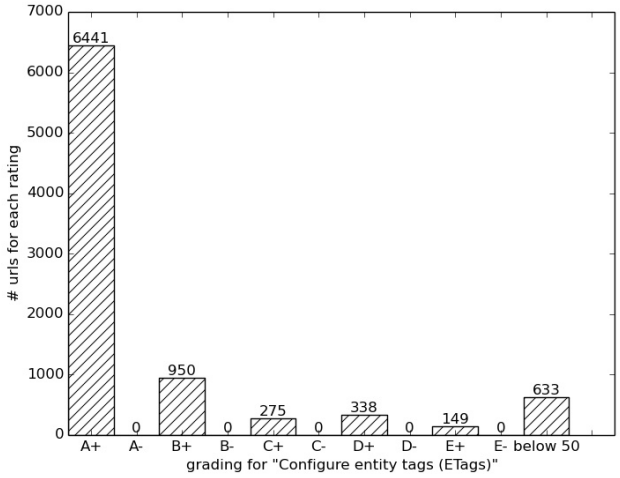


(a) 'Compress Components with gzip' rule with PageSpeed

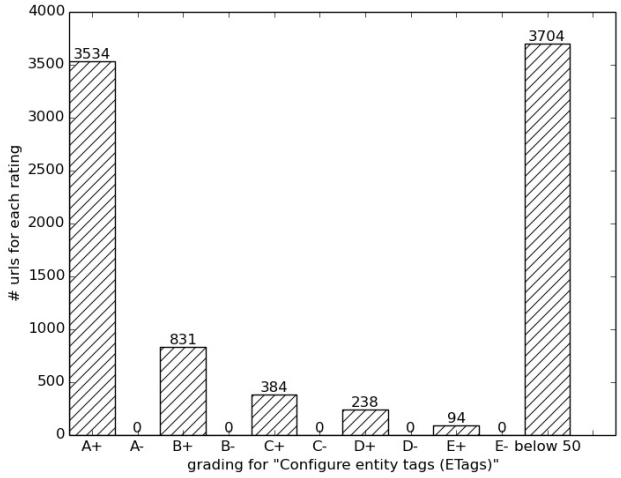


(b) 'Compress Components with gzip' rule without PageSpeed

Fig. 12: Comparing 'Compress Components with gzip' rule

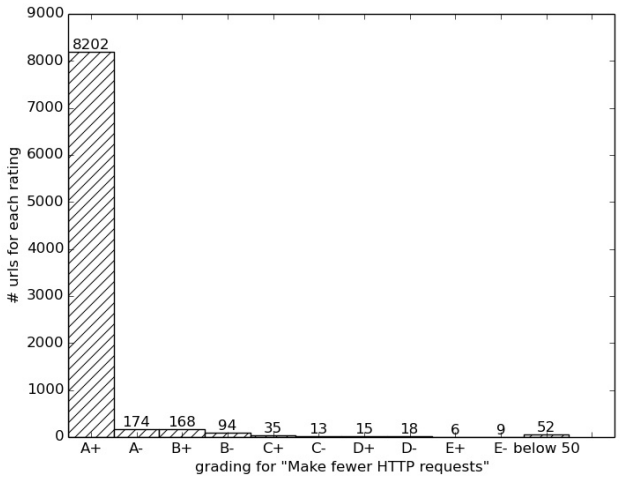


(a) 'Configure entity tags (ETags)' rule with PageSpeed

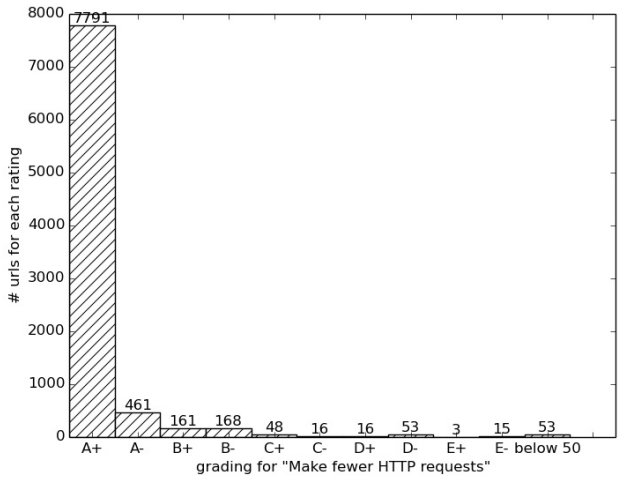


(b) 'Configure entity tags (ETags)' rule without PageSpeed

Fig. 13: Comparing 'Configure entity tags (ETags)' rule

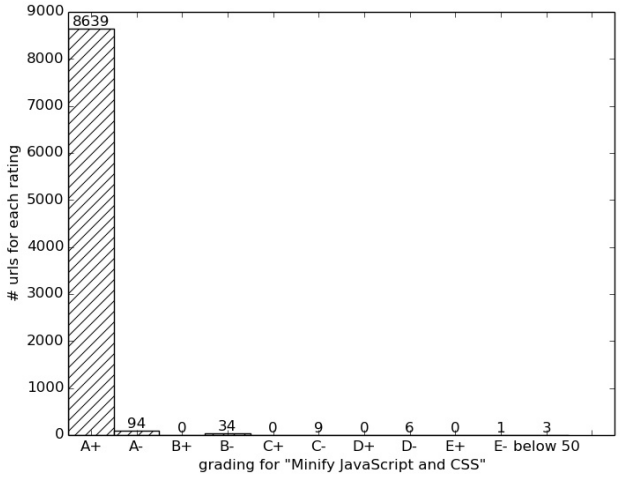


(a) 'Make fewer HTTP requests' rule with PageSpeed

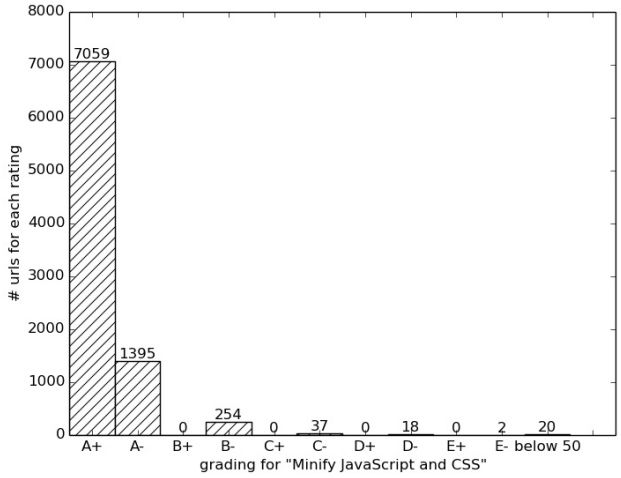


(b) 'Make fewer HTTP requests' rule with PageSpeed

Fig. 14: Comparing "Make fewer HTTP requests' rule



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



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

Fig. 15: Comparing 'Minify JavaScript and CSS' rule

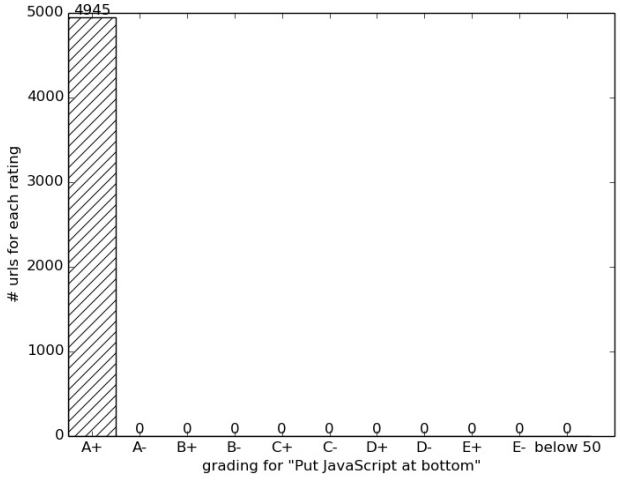


Fig. 3: Number of urls versus Put JavaScript at bottom

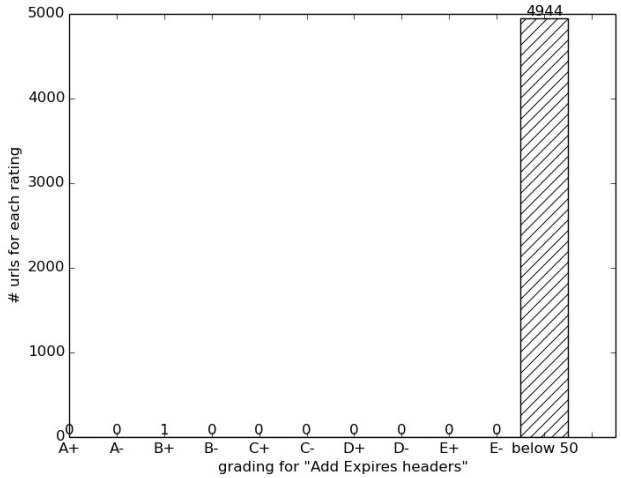


Fig. 5: Number of urls versus Add Expires Headers

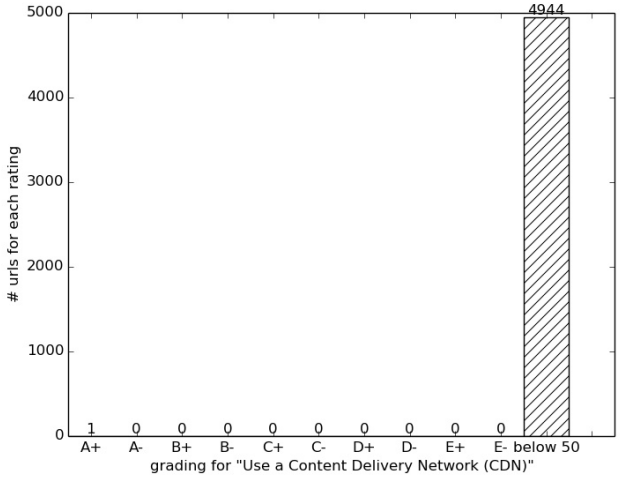


Fig. 4: Number of urls versus Use a Content Delivery Network (CDN)

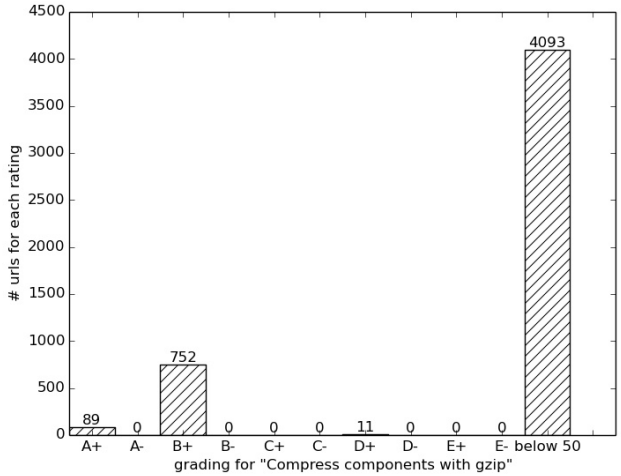


Fig. 6: Number of urls versus Compress components with gzip

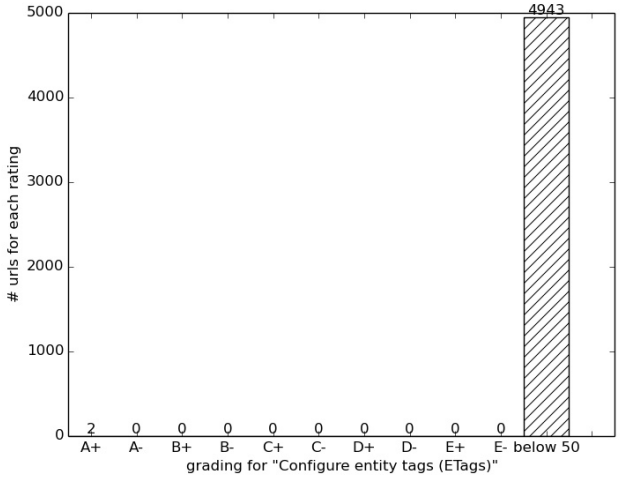


Fig. 7: Number of urls versus Configure entity tags (ETags)

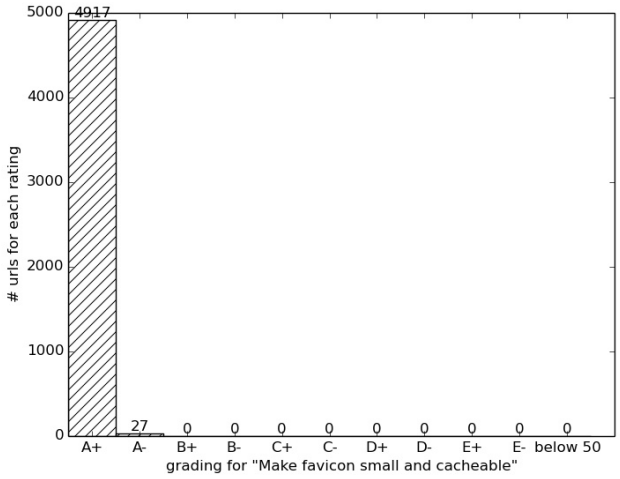


Fig. 8: Number of urls versus Make favicon small and cacheable

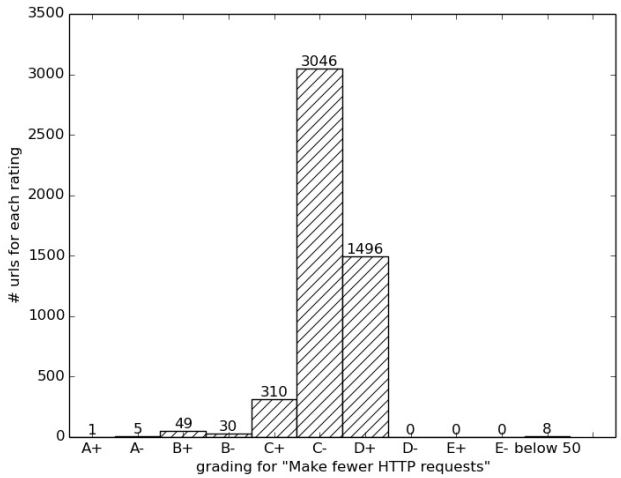


Fig. 9: Make fewer HTTP requests Versus Number of urls