Analysing Page Load Times by Optimising and Separating Unique Functions in Internal Web pages

Aayush Aayron Deo
Computer Science, NYUAD
aad598@nyu.edu

Advised by: Yasir Zaki

ABSTRACT

JavaScript the language of the web, is very widespread. About 95% of websites running some JavaScript. Using a lot of JavaScript can slow down the website due to large files being sent over the network, and the need to interpret lots of lines of code. Some of the JavaScript maybe non-essential to the functionality of the website, or may never be used due to not being triggered by user actions and as such can be removed, reducing the overall bundle size, and lines of code to interpret resulting in an increase in load times and performance. I present the analysis metrics of pages whose JavaScript files were optimised through detection and elimination of deadcode using Muzeel and detect functions that are extra in an internal compared to all the functions present in the JavaScript of the homepage. Putting these extra function a file and referring to this file benefits performance if the internal page does not reference any other JavaScript resource that is not referenced by the homepage.

KEYWORDS

internal pages, optimisation, JavaScript, load times, deadcode, extra functions, unique functions, functions

Reference Format:
Aayush Aayron Deo. 2022. Analysing Page Load Times by Optimising and Separating Unique Functions in Internal Web pages. In *NYUAD Capstone Project 2 Reports, Spring 2022, Abu Dhabi, UAE.* 5 pages.

This report is submitted to NYUAD's capstone repository in fulfillment of NYUAD's Computer Science major graduation requirements.

جامعة نيويورك أبوظبي
NYU ABU DHABI

Capstone Project 2, Spring 2022, Abu Dhabi, UAE

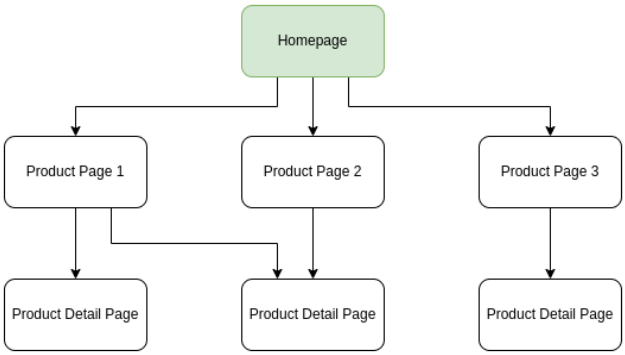

Figure 1: A site map of a website that shows path to internal pages

1 INTRODUCTION

JavaScript is the language of web and about 95% of websites have some JavaScript included. This JavaScript code has multiple purposes. For example, making network requests, enabling animations, adding interactivity, checking authentication, conditional rendering. While having JavaScript on websites enhances the user experience, it comes with the cost of having to transfer multiple and at times large JavaScript files over the network and interpreting the code to make it functional on the website. This may not be an issue on many mid to higher tier mobiles devices; however, on lower tier mobile devices with limited RAM and CPU power this can significantly increase the time to interactivity of a web page, this is all on top of the time it takes to download the JavaScript files over a slow network.

Our aim is to detect, eliminate JavaScript dead code home page and internal web pages using Muzeel and also to separate the detect the functions in the internal which is the unique to the internal page when compared to the homepage. An internal page is defined as any page of a website that can be reachable by following a path, either directly or through another page from the homepage of a website.

By separating the extra functions a separate file and by leveraging browser caching, we could gain improvements to page load times. In this paper I discuss the methodology, the analysis and testing techniques used, the results of the study and works related to the paper.

2 METHODOLOGY

2.1 Hispar List

The Hispar[2] is a list published by weekly by Duke University that contains the list of top webpages as compared to the top domains as published by other web ranking services like the Alexa and Tranco lists, which only published the top domains. The Hispar list is much more applicable to our study as it list the internal pages of a site as well as the homepage of the internal webpage. This is useful so that we can access the homepage assets as well as the internal page assets. Not only that but the Hispar list also ranks the pages based Alexa ranking. We leveraged this by conducting the study on the top 3 internal pages and its corresponding homepage.

We selected the pages such that when a page is accessed using the url from the Hispar, it goes directly to the page and does not redirect. This is important for Muzeel to work correctly.

2.2 Scraping and Asset Caching

After the list of internal pages and its corresponding homepages were selected, these sites were scraping using the MITM proxy which seeing all requests that page is making and would intercept, downloading and cache these resources for use with Muzeel later. During the scraping, MITM proxy would see a request, intercept the request, log the relevant and also download and store the requested resources on the server to be served later.

2.3 Muzeel

Muzeel is a tool that was developed by Tofunmi as part of his capstone to detect deadcode in JavaScript files. The Muzeel was extended from LacunaJS. Muzeel detects the deadcode by the means of a directed graph. Muzeel would first go through JavaScript files and adds a node for every function that is encounters in the JavaScript file. After this stage, starting from the global scope, Muzeel would add a directed edge from function A to function B where function A calls function B. After this graph is made, Muzeel looks for nodes that have an indegree of zero which means that the respective node was not called by any other function. The only expection to this rule is the global node where the execution starts. These functions are declared as deadcode and the function body for each of these functions are removed. Muzeel removes function body as opposed to the functions declaration because there maybe instance where these functions are values

```
1   // Function Declaration
2   function sumA(a, b) {
3       return a + b
4   }
5
6   // function expression
7   const sumB = function(a, b) {
8       return a + b
9   }
10
11  // arrow function
12  const sumC = (a, b) => {
13      return a + b
14  }
```

Figure 2: Screenshot showing three types of functions in JavaScript

to keys in a JavaScript object. If the interpreter does not see this function, it would throw an error and would stop executing. Removing the function body has the same effect as not having the function itself.

2.4 Extra Functions

There are three type of JavaScript functions: function declarations, function expression and arrow function as seen in Figure 2. To detect extra unique functions for an internal page, all the JavaScript files [which has been optimised by the Muzeel in the previous] as parsed using Esprima. Esprima would list all the JavaScript components for the file. We repeat the parsing for all JavaScript files that are used by the homepage. After both sets of JavaScript files are parsed by the Esprima, we extract the names of all functions that are accessible from a global scope and keep track of them by the type of page [internal page or homepage]. Using the list all the function names, we compare the list of internal page functions with the list of the homepage function and extract the extra function in the internal page. Extra functions are defined as those function which are present in the set of internal page function but not present in set of homepage functions.

After the extra functions are identified, these extra functions are written in the form a function declaration and put in a separate file. After this, HTML page of the internal page is injected with the script tag that refers to the extra JavaScript functions so that this resource can be extracted and served by our proxy when running the tests.

3 ANALYSIS

We conducted two different types of analysis. The first was we ran the web page test on three different scenarios. Another test we ran was to we compare the structural similarity of the two pages using QLUE.

3.1 Webpage Test

Webpage test is a tool that we used to get the metric of the webpages. We ran webpage tests for three different scenarios:

(1) Visiting original internal page
(2) Visiting internal page whose JavaScript has been optimised by Muzeel and includes extra functions for that page
(3) Visting landing page whose JavaScript has been optimised by Muzeel then visiting the internal page whose JavaScript has been optimised by Muzeel and includes the extras function for that page

The above tests were five time for each of the 14 internal pages using a proxy that would serve the requested resource through our proxy instead of requesting it from the original source. Due to the dynamic and the templated nature of the most webpages today, webpages may change depending on when it was requested. Given that all the pages were scraped and all the resources for the page were available on our server, we used a proxy to serve these resources. This would ensure that the same resources will be served.

3.2 QLUE

QLUE is tool that uses tools such as skiimage to break up the screenshots of two pages into smaller components and would analyse then based on these components. QLUE would output a score between 0 and 1 for a page of webpages. In our test we ran QLUE for screenshot of the original page compared to the screenshot of the internal pages who JavaScript has been optimised based by Muzeel and includes the extra functions for that internal site. The screenshots generated were for the entire length of the webpage.

4 RESULTS

4.1 Webpage Test

The webpage test reported the metrics that relate to the loading of the pages. As mentioned above the 3 testing scenarios were repeated 5 times for each of the 14 internal page and their corresponding homepages where possible. To normalise the data from the 5 runs of a single test on a page, the data point was sorted and the median value was taken for the test. Three different combinations of the tests ran were analysed. These tests were:

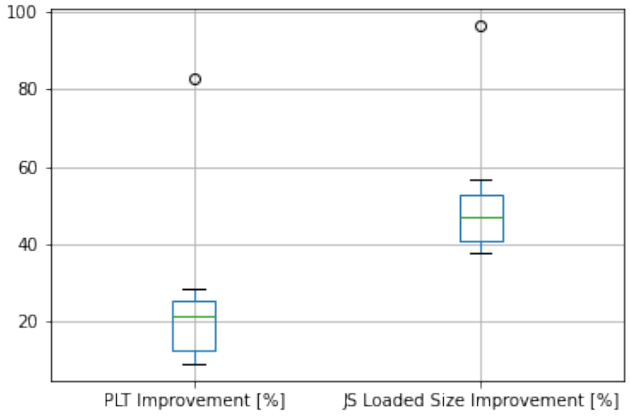

Figure 3: Box plot comparison A

(1) Comparison A - Metrics for visiting original landing page vs the metrics for visiting internal page with Muzeel optimised and extra JavaScript functions
(2) Comparison B - Metrics for visiting original landing page vs the metrics for visiting the landing page with Muzeel optimised and then visiting internal page with Muzeel optimised and extra JavaScript functions
(3) Comparison C - Metrics for visiting internal page with Muzeel optimised and extra JavaScript functions vs the metrics for visiting the landing page with Muzeel optimised and then visiting internal page with Muzeel optimised and extra JavaScript functions

4.1.1 Comparison A. The purpose of this test was to determine if there were significant improvements in page load times and size of JavaScript requested. This showed that on average the internal page [optimised with Muzeel and containing extra JavaScript functions] loaded 24.14% faster compared to the original and there was on average a decrease of the 50.4% bytes loaded when comparing the size of the JavaScript file. This improvement came about due to the use of the Muzeel which remove the JavaScript deadcode and reduced size of the JavaScript file to be loaded.

4.1.2 Comparison B. The purpose of this test was to determine if there were significant improvements in page load times and size of JavaScript requested. This showed that on average the internal page [optimised with Muzeel and containing extra JavaScript functions] visited after visiting the landing page [with JavaScript optimised by Muzeel] loaded 20.12% faster compared to the original and there was on average a decrease of the 48.16% bytes loaded when comparing the size of the JavaScript file. This improvement came about due to the use of the Muzeel which remove the JavaScript deadcode and reduced size of the JavaScript file to be loaded.

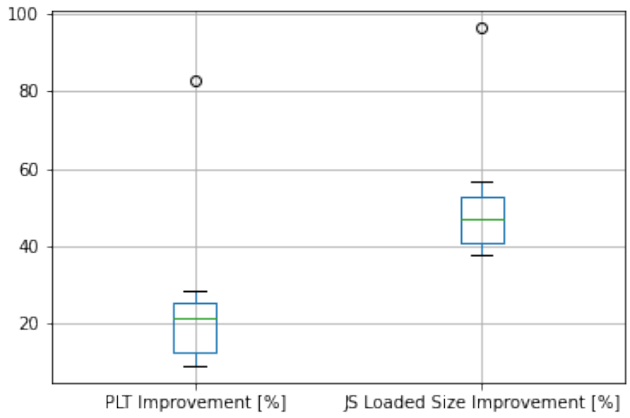

Figure 4: Box plot comparison B

4.1.3 Comparison C. Out of the 14 web pages that the test was run on the only 5 web pages [internal page visited after landing] loaded faster as compared to visiting the internal page directly. These pages loaded on average about 23.02% faster and saw a decrease of 8.84% bytes loaded. This improvement came out due to the browser caching the resources that it had already been downloaded. As for the other 9 internal sites, the internal pages visited after landing loaded slower as there were resource origin urls that the web browser had not previous encountered as such the browser had to download these resources which increased the load time.

4.2 QLUE

After running QLUE on 14 internal pages, QLUE returned a score of 1 for the 11 out of the 14 pages. A score of 1 means that the original internal page [without modification] and the modified internal page [whose JavaScript files had been optimised by Muzeel and had extra functions] were completely similar. Out of the 3 that were not accounted for of the pages loaded with a blank screen and as such the test failed for the page, while for the other two pages the test was inconclusive.

5 RELATED WORKS

Research conducted by Sohaib et al[1] using mobile phone statistics of about 0.5 millions smartphones from Pakistan, revealed that 57.4% of mobile phones had CPU speeds in the range of about 500-1000MHz whereas only 0.485% of phones had more than 1GB of memory. Smart phones with such specifications have limited capability and would not be able run large JavaScript files. Research by Usama et al[4] explored the causes that contributed to the sub optimal performance of web pages by longitudinal cross-analysis of resource profiles from a large social network in five regions. Usama et al proposed WebMedic an approach to JavaScript analysis and optimisation that traded off less critical functionality of a web page to directly address memory and performance problems. The paper reports that WebMedic can reduce JavaScript memory by 80% while giving up only 20% of the functionality. This would be helpful for lower end smartphones.

The paper by Neils et al[5] proposed LacunaJS, a tool for JavaScript deadcode elimination, that uses several methods for the analysis of JavaScript code for dead code before removal. LacunaJS creates a graph of function call graph, then using one of three analysis techniques [static, dynamic or WALA-based extensions] analyzes, the function calls. In the graph, each node is a function and during the analysis an edge is added from the calling function to the called functions. At the end of the analysis, dead code is identified as nodes which have no inward edges, this means this function is not called by any other function and the function body can be removed without any repercussions. Another approach to the JavaScript optimisation involves the use of the use of algorithms like genetic programming and genetic improvement as outlined by Fabio et al[3].

The approach by Fabio et al, reported size reductions in JavaScript bundles from 6% to 17% and a 40% reduction in unused code. This approach could be used in conjuction with LacunaJS outlined by Neils et al to potentially increase the level of dead code elimination.

6 CONCLUSION

In conclusion, Muzeel on the JavaScript files of a webpage can reduce the total size of the JavaScript to upto 50% while reducing loads times by upto 24%. In terms of internal pages, if both the home page and internal page references the same amount of JavaScript resources, by leveraging caching [if the landing page was visited before the internal page] load times could be reduced by upto 23% and the amount of the JavaScript loaded can be reduced by upto 8%. However, if it is the same that the internal reference JavaScript resources that were present in the homepage when the change in metrics are not beneficial but rather can be more detrimental.

Having these improvement significantly improve access to the internet in the places with slower internets as this would mean that pages would load faster due to smaller files being downloaded as well the JavaScript engine having to parse smaller files.

REFERENCES

[1] Sohaib Ahmad, Abdul Lateef Haamid, Zafar Ayyub Qazi, Zhenyu Zhou, Theophilus Benson, and Ihsan Ayyub Qazi. 2016. A View from the Other Side: Understanding Mobile Phone Characteristics in the Developing World. In *Proceedings of the 2016 Internet Measurement Conference* (Santa Monica, California, USA) *(IMC '16)*. Association for Computing Machinery, New York, NY, USA, 319–325.

https://doi.org/10.1145/2987443.2987470

[2] Waqar Aqeel, Balakrishnan Chandrasekaran, Anja Feldmann, and Bruce M. Maggs. 2020. On Landing and Internal Web Pages: The Strange Case of Jekyll and Hyde in Web Performance Measurement. In *Proceedings of the ACM Internet Measurement Conference* (Virtual Event, USA) *(IMC '20)*. Association for Computing Machinery, New York, NY, USA, 680–695. https://doi.org/10.1145/3419394.3423626

[3] Fabio Farzat, Marcio Oliveira Barros, and Guilherme H. Travassos. 2019. Evolving JavaScript code to reduce load time. *IEEE Transactions on Software Engineering* (2019), 1–1. https://doi.org/10.1109/TSE.2019.2928293

[4] Usama Naseer, Theophilus A. Benson, and Ravi Netravali. 2021. WebMedic: Disentangling the Memory-Functionality Tension for the Next Billion Mobile Web Users. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications* (Virtual, United Kingdom) *(HotMobile '21)*. Association for Computing Machinery, New York, NY, USA, 71–77. https://doi.org/10.1145/3446382.3448652

[5] Niels Groot Obbink, Ivano Malavolta, Gian Luca Scoccia, and Patricia Lago. 2018. An extensible approach for taming the challenges of JavaScript dead code elimination. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 291–401. https://doi.org/10.1109/SANER.2018.8330226