Poster: Rapid Web-Page Loading with Preload Scanner

Jemin Ahn
ahnjemin@hanyang.ac.kr
Hanyang University

Jiwoong Won
jiwoongwon@hanyang.ac.kr
Hanyang University

Kyungtae Kang
ktkang@hanyang.ac.kr
Dept. AAI, Hanyang University

ABSTRACT

Despite being continuously advanced, conventional web page loading processes remain inefficient and prevent network devices from fully utilizing their computing capabilities. This study investigates the primary source of this inefficiency, known as script blocking, which occurs owing to browser pipelining. An advanced web page loading process is then proposed, in which the Preload Scanner in Chromium is modified such that it can identify correlations between web objects and prerequests associated with them. Experimental testing with example websites demonstrates an improvement in overall web page load times.

ACM Reference Format:
Jemin Ahn, Jiwoong Won, and Kyungtae Kang. 2021. Poster: Rapid Web-Page Loading with Preload Scanner. In *IMC '21: The 2021 Internet Measurement Conference, November 02–04, 2021, Virtual Event*. ACM, 2 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

When a HTML web page is rendered in browsers, the browser creates a Document Object Model (DOM) tree, but it is sometimes hindered because of the script blocking problem [3]. Owing to this problem, the sequential requests for loading script files, which should be executed in sequence, may reach the browser out of order. Moreover, modern web pages are becoming increasingly complicated, requiring a significant amount of execution time to evaluate the script and cascading style sheets (CSS), and to render the page. To resolve these problems, web browsers provide a preload function using a Preload Scanner, which scans and prerequests certain network resources before parsing the HTML document. This has been shown to be effective for improving web page loading performance. However, the link tag prerequests created by the Preload Scanner are given higher priority than script tag ones. This trait occasionally delays the loading of the script, compounding the script blocking problem.

In this study, the internal structure of the Preload Scanner provided by the Chromium web browser was modified such that it sends script prerequests with the same priority as prerequested link tags to alleviate the script blocking problem by loading the script data faster. Experiments with example websites demonstrate an improvement in the page-loading performance, along with insights about the correlation between prerequest priority and HTML object loading.

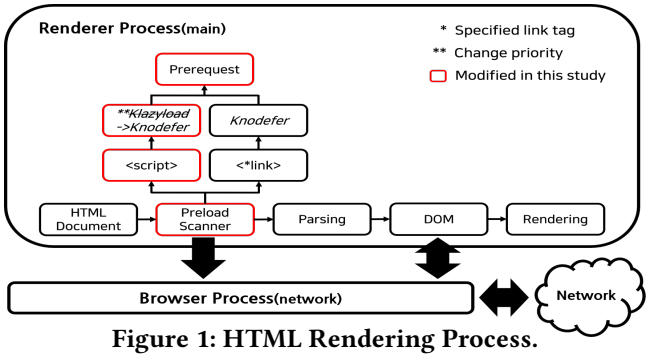

Figure 1: HTML Rendering Process.

2 BACKGROUND

2.1 Script Blocking Problem

Modern web browsers use pipelining to improve browser performance; in Chromium, pipelined tasks are executed for every other process (Figure 1). Network transmissions and script evaluations are executed simultaneously while the DOM tree is being created. Some script files in the loaded HTML document are related to each other, and these files must be executed sequentially to ensure that the result of the DOM tree value is consistent. To maintain this order, DOM generation is stopped when the prior scripts are not loaded or evaluated, even if other scripts are already loaded. This interruption of DOM-creation degrades browser performance.

2.2 Preload Scanner

Pausing the parser whenever a script is encountered means that every loaded script delays the rest of the resources linked within the HTML document. The Preload Scanner in Chromium allows the necessary files to continue to be downloaded while the loaded files are being evaluated. It is a class that preloads objects by creating prerequests for script

IMC '21, November 02–04, 2021, Virtual Event

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

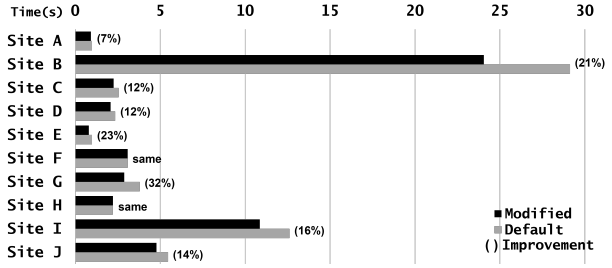

Figure 2: Average DOM-creation Time.

tags and specified link tags before the DOM tree generation begins. While script tags are converted using the *klazyload* parameter, link tags to be preloaded can be created using the markup and HTTP header specification. This implies that not all link tag data are subject to preload processing, because link tags do not cause an interruption in DOM parsing.

Generally, files related to styles such as fonts are preloaded by the HTML document writer, but other data that the writer needs to load quickly can be preloaded to improve web page response. Link tags have several preload attribute specifications [2]. If the preload attributes for the link tags are specified, the Preload Scanner makes prerequests with the fetch parameter *knodefer*. These prerequests have higher priority than script tags prerequest parameter *klazyload*.

3 EVALUATION

Here, the Preload Scanner was modified in Chromium. During the preload process, script tags are treated as *klazyload* attributes, which causes them to have lower priority than prerequested link tags. However, script tags typically affect the DOM-creation process of the requested web page. By contrast, link tags are usually used to load files or links and do not cause the script blocking problem. To quickly display the requested web page and more efficiently solve the blocking problem, it is preferable to load the scripts faster or with the same priority as the prerequested link tags.

The computational power of existing machines may be adequate to improve page load time (*PLT*) via simple modifications. This is because the blocking problem is caused by an inefficient loading process, not a lack of performance. Here, HTML script tags were modified such that they were processed with the *knodefer* attribute, giving them the same priority as the prerequested link tags. This reduces blocking time by expediting the entire script loading process. Expectedly, this leads to higher network traffic during the initial web page load stage but helps to alleviate the blocking problem that occurs after loading. Ten Widely accessed websites were selected from the Alexa top 50 global websites [1], and the load test was performed 50 times using the Chromium browser on Ubuntu 16.04 LTS with no cache condition.

The time duration between the completion of initial-HTML-document loading and completion of DOM-creation was measured to quantify the effect of changed script loading. Figure 2 shows the changes in DOM-creation time for 10 websites, comparing the default browser to the proposed modified browser. All the websites exhibit improved DOM-creation performance with faster script loading except for websites F and H. As shown in Table 1, a better improvement is obtained with the increasing number of prerequests that are changed to *knodefer*.

Table 1: Changed Prerequests for the Websites.

Site	The number of prerequest	Changed prerequest	Improvement	Average improvement
F	16	0	-	
H	18	0	-	
A	12	1	7%	
C	22	2	12%	
D	72	1	12%	13.6%
E	13	2	23%	
J	21	4	14%	
B	22	10	21%	
G	34	10	32%	23.5%
I	24	11	16%	

The first group with no changed prerequest exhibits the same performance because there are no prerequests for script loading; Prerequests are unaltered in the modified browser. The second group has less than four changed prerequests, and they show a 13.6% average improvement. The last group has more than 10 changed prerequests, and they show a 23.5% average improvement in the modified browser. Nonetheless, the case of websites E and I demonstrates that the number of changed prerequests is not always proportional to performance improvement; in addition, when considering performance differences in the same group, the total number of prerequests and HTML document environments matters.

4 CONCLUSION

We demonstrated that the design of advanced pre-load function (for reducing DOM-creation time) is feasible to achieve considering the enhanced performance of modern devices, and it can help to overcome the script blocking problem.

ACKNOWLEDGMENTS

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No.2020-0-01343) and the National Research Foundation of Korea (NRF) grant funded ty the Korean government (MSIT) (No.1711117678). First two authors contributed equally.

REFERENCES

[1] Alexa 2020. Top sites in South Korea. https://www.alexa.com/topsites/countries/KR.
[2] W3C 2019. Preload. https://www.w3.org/TR/preload/.
[3] Xiao Sophia Wang, Arvind Krishnamurthy, and David Wetherall. 2016. Speeding up Web Page Loads with Shandian. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA, 109–122.