# In depth analysis of mobile web performance and computation

Ayush Goel Matthew Furlong Hyun Jong Lee

## 1 Introduction

Page Load Time (PLT) of a website is a key performance metric that significantly impacts user-experience, as pointed out by many recent studies from both academia and industry [**?**, **?**]. User-experience in web browsing is directly correlated to companies' revenues: Amazon shows that reducing 100ms in PLT results in 1 percent revenue increase and Shopzilla reports that improving PLT from 6 to 1.2 seconds increased the revenue by 12 percent [**?**].

There have been many prior works in improving webpage's PLT of mobile devices, ranging from offloading computation and network tasks to proxies [**?**, **?**, **?**] to reprioritising requests at client side by letting the client itself discover all resources on a page [**?**, **?**]. It is worthwhile to note that existing works attempt to surrogate web-tasks of mobile devices from resource-rich server environment [**?**].

The end-to-end PLT for many webpages is far from the ideal: an order of tens of seconds on mobile devices [**?**] and order of seconds on stationary desktops. Many existing solutions often require server-side modification, which strongly discourages content providers to use new solution. Whereas a client side solution would be agnostic of the content provider/server that is being used to render the web pages and optimize page load times for all web pages alike. Most of the existing work on client side talks about efficient ways to optimise web cache [**?**]. Prior work shows how computation latency is the driving factor behind large web page load time, as compared to network latency [**?**].

In this work, we propose PLT optimization technique that caches output from previous code execution of a webpage (e.g., Javascript, inline HTML, css) on mobile devices to reduce user-perceiving PLT in exchange of prevalent mass-storage. Prior techniques have gone as far as caching the compiled code either on the client side or on the server side to save on the compilation time when the web content remains unchanged [**?**]. We take this a step further, and cache the output of the execution of all the code on a web page. ( Note that I use the word code, to distinguish it from other components of a webpage which include layout and data). Recent work has shown that most of the webpages remain unchanged over a large period of time. For content-rich pages, the amount of updates vary across Web pages. In the best (worst) case, 20% (75%) of the HTML page is changed over a month. Most changes are made to data (e.g., links to images, titles) while little is made to layout and code [**?**]. This implies that most of the code output could be reused, essentially eliminating code execution time from the critical path of a web page load. This would bring down the entire page load time to the time taken in rendering/painting the layout. Caching the computation as a technique to optimize the execution time has already been explored at a data center level [**?**] and it has shown tremendous improvement with more than 35% job benefiting from caching. We are trying to apply a similar technique however on the client side at a browser level.

## 2 Motivation

Major web browsers like Chrome, firefox, Safari have recenlty invested a lot of resources, time and energy in improving the web performance on the mobile, specially by targetting the network usage. So much so that the network now comprises less than 30% [] of the total critical path for an average page load on the mobile device. This includes caching almost 95% of the resource that is fetched from the server [], dns presolution, dns caching, tcp reconnect etc. Chrome released a paper last year talking about how improved caching algorithms, despite having significant improvements on the desktop, don't have the same proportionate imrpovements on the mobile. This is primarily attributed to the fact that computation comprises more than 65% of the critical path during a page load. This presses the need to further optimise the compuatation time.

During the Chrome dev summit this year, their team announced the latest improvements they have made in their browser to improve the page load time. Interstingly most of their work talks about improving the compilation and parsing time by introducing compile and parser cache. Recent studies [] still report that the median page load time for a mobile website is about 14 seconds. Research [] shows that a user will only wait for 3 seconds before abandoning a web site if it shows no response at all. A lot of prior work [] has been done to compare the page load times on mobile vs desktop, and recent results uptil last year claim that despite the increasing com-

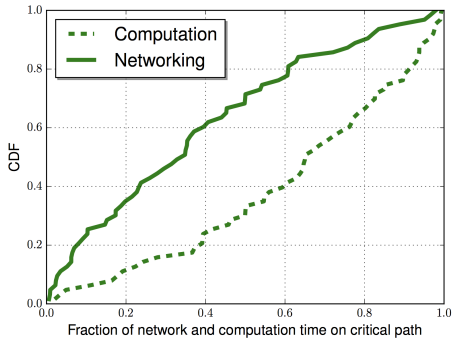


Figure 1: **Runtime information on mobile devices**

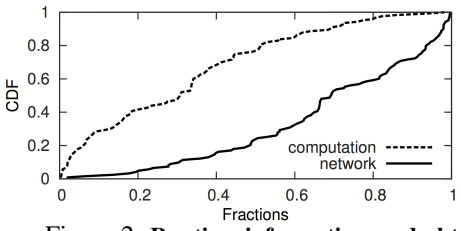


Figure 2: **Runtime information on desktops**

pute resources in mobile devices the computation time on mobile is significanlty higher than their desktop counterparts. Our experiments on the most popular news and sports websites on the latest mobile hardware and the latest chrome verison reveal that despite these recent efforts, scripting still takes a significantly more amount of time as compared to the other components of the total computation time. We break down computation into four categories: scripting, loading, paitnint, rendering. and observe that scripting essentially takes more than 70% of the total computation time which is more than all the other categories combined. This makes it all the more important to do an in depth of analysis of the computation time to clearly understand where exactly is the time being spent.

# 3 Design

We propose a new technique to imrpove the page load time by reducing the javscript time. In order to do this we are trying to build a new caching framework for the modern web browsers specifically, Chrome since it accounts for about 50% of the market share in terms of browser usage. Our caching framework will store the javascript execution result. This can mean a lot of things due to the dynamic nature javascript. Most of the times it is supposed to be the return value of the javascript functions. At other times it can be a modified DOM structure or just some intermediate result which is further processed by other javascript, later down the execution timeline. The expiry of this javascript exectution cache is supposed to be same as the expire of the javascript source

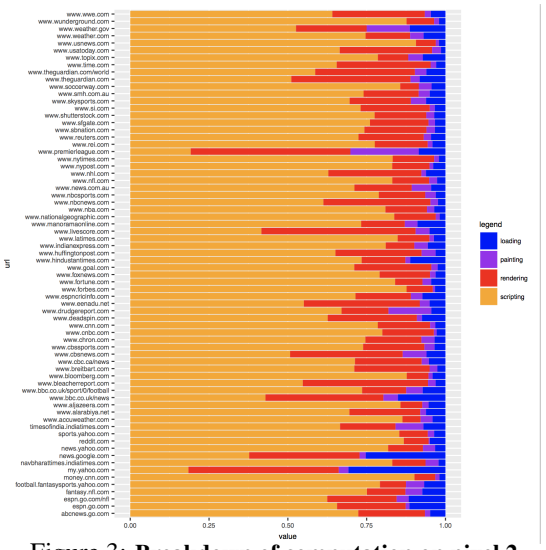


Figure 3: **Breakdown of computation on pixel 2**

cache. There are a lot of caveats to this approach, and in our work we try to essentially explore all of these. The biggest challenge with a new caching framework are the actual modifications to the current browser's code in order to evaluate the efficacy of our caching frameowork. Since a lot of browsers already implement caching at the javascript runtime level, such as compile and parses cache, a lot of this architecure can be borrowed for the execution cache as well. Another possible challenge can be the memory overhead. Most of the popular websites which spend about 70% of their time on javascript execution, run about 1000s of javascript functions. Saving the output of all of these functions can add an extra overhead on the current browsers.

# 4 Progress

As a preeliminary step in research, we first established a corpus of the top 75 news and sports websites on which we woudl run most of our experiments in order to cater to the most popular and compute intensive websites. These websites are taken from the Alexa top website list []. We ran all our experiments on Macbook Pro 2017, Google Pixel 2, and Moto G5 to contrast performance numbers between the three most famous devices with drastically different compute powers. We used chrome v 61 to run all our experiments both on the desktop and the mobile devices. We leveraged chrome developer tools in order to capture the runtime traces both for network and compute. We then analysed these runtime traces to draw insight into the critical path of the website, the total time being spent on the compute vs the total time being spent on the network, and most importantly the finer level breakdown of the computation time to understand the true nature of computation on mobile devices.

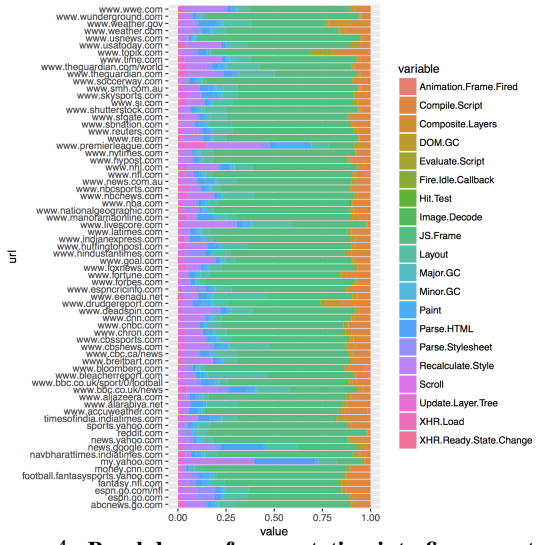


Figure 4: **Breakdown of computation into finer events on pixel 2**

We categorised computation time into four categorires, scripting, loading, rendering and painting. Scripting is the total time being spent on compiling javascript, evaluating and executing it. Loading is the parsing of html and css, which happens immediately after the payload for the network requests are received by the browser. Loading essentially takes these payload objects and parses them and converts them to a DOM tree. Once the DOM tree is built, the rendering engine kicks in which starts to convert this DOM tree into a render tree, which contains the exact coordinates and the shape of each of the DOM node. This is what rendering time comprises of. Painting time is the time taken to process the render tree, and actually convert each pixel into a bitmap. Figure 3 shows the computation break down for these first level of categories for Google pixel 2. We further break down this time into the most finest level events which are returned by Google Chrome's trace and then group them by their event name.

These results are extremely coherent with our design of implementing a javsvcript chrome caching mechanism and show the possibility of a large improvement in the total page load due to the high percentage share of execution time as seen in Figure 4.

Recently Chrome in their 2017 dev summit talked about the various optimisation techniques they have employed in the latest chrome browser to improve the total page load time. We did a comparison of the total page load with and without chrome's optimisation to see the improvements. As we can see, there has been a significant reduction in the overall scripting time. This is primarily due to the introduction of compiler and parser cache. We further broke down this computation and compared the exact compile time across different optimisation settings. We first captured trace for the top 75 websites with the chrome compile and parse optimization disabled and then again with the optimization enabled. As shown in Figure 6 to our suprise, there was no reduction in the compile time, as opposed to the claim by Chrome quoting a 40% decrease in the overall compile time. All of our work will be on top of Chrome's latest changes to ensure compatibility and relevance of our work.

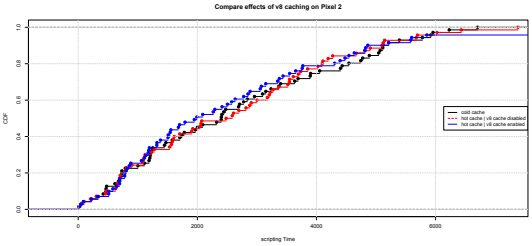


Figure 5: **Comparison of total scripting time with and without chrome's optimisations**

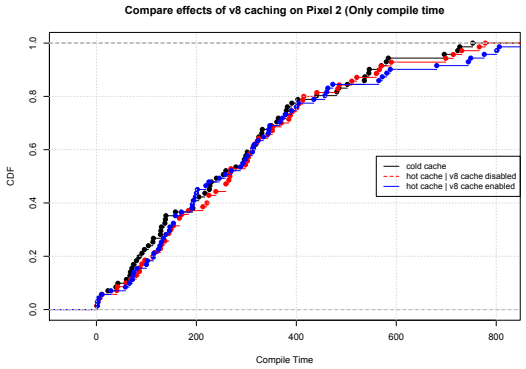


Figure 6: **Comparison of total compiling time with and without chrome's optimisations**

# 5 Related Work

Work on improving web performance has been going for more than two decades now. Prior work has focused on various components of the overall page load time from remodifying the source code of the webpage itself, to optimizing the network component of the over all execution time and more recently some work has been in improving the computatio time latency.

Erman et all [] has shown that unline desktop browsers, optimizations such as SPDY/HTTP@ does not improve performance of Web pages on mobile browsers. They show that this is because of the negative interactions between the cellular state machine and the transport protocol. SImilary Qian et al [**?**] show that caching does not provide page load improvements for mobile browsers. Google already released a paper last year [] which talks of how there is very little improvement to the

overall page load time despite significant improvements in the caching hit rate.

Many of the research on explicitly improving mobile browser performance has seen mixed results. FLywheel [] is Google's compression proxy that compressses web content to significanlty reduce the use of expensive cellular data. The authots note that whie Flywheel succeeds in reducong the data usage, its effect on page load performance is more mixed; it helps performance of certain pages and hurts performance of others. Flexiweb [] is built over Google's compression proxy to ensure that the proxy does not hurt page load times. But FlexiWeb is not designed to explicitly improve page load performance. Wang et al [**?**] show that speculative loading in one of only client only approaches that can improve mobile browser performance. However, speculative loading requires knowledge of what objects are likely to be requested by the user.

Other research works have looked at metrics orthognal to the page load time metric. Parcel [] uses a proxy approach to divide the page load process between the mobile device and the proxy. Becayse Parcel is a network appraoch, the evaluations are largely with respect to reduction to network latencies. Klotski [] focusses on increasing the number of objects rendered in the first 5 seconds to improve the user quality of experience.

Other client side imporvements reduce enrge usage and computational delays using parallel browsers [] and improved hardware [] By improving the parallelization for necessary page load tasks (eg: rendering) these systems reduce energy usage and have positive impact on page load times.

While there has been several recent efforts on improving mobile browser performance they have not been uniformly successful.

## References