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 [2, 3]. 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 their revenue by 12 percent [8].

There have been many prior works in improving the PLT of mobile devices, ranging from offloading computation and network tasks to proxies [13, 18, 22] to reprioritising requests at client side by letting the client itself discover all resources on a page [4, 12]. It is worthwhile to note that existing works attempt to surrogate web-tasks of mobile devices from resource-rich server environment [15].

The end-to-end PLT for many webpages is far from the ideal: an order of tens of seconds on mobile devices [21] and on the order of seconds for stationary desktops. Many existing solutions often require server-side modification, which strongly discourages content providers to use these new solutions. However, a client side solution would be agnostic of the content provider/server that is used to render the web pages and can therefore optimize page load times for all web pages alike. Most of the existing work on client side optimiztions focuses on efficient ways to optimise web cache [23]. Prior work shows that computation latency is the driving factor behind slow page load time for mobile devices, as compared to newtork latency. [20].

In this work, we propose a novel 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-perceived PLT at the small cost of increased storage requirements. 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 [23]. 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 change is made to the layout and code [23]. 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 and painting the layout. Caching the computation as a technique to optimize the execution time has already been explored at a data center level [7] and it has shown tremendous improvement with more than 35% of jobs benefiting from caching. We are trying to apply a similar technique on the mobile client's browser.

2 Motivation

Major web browsers like Chrome, Firefox, and Safari have recenlty invested a lot of resources, time and energy into improving web performance on mobile devices, specifically by targeting the network usage. However, the network now comprises less than 30% [11] of the total critical path for an average page load on a mobile device. This includes caching almost 95% of the resources that are fetched from the server [20], dns presolution, dns caching, tcp reconnect etc. Chrome released a paper last year showing how improved caching algorithms, despite having significant improvements on the desktop, don't have the same proportionate imrpovements on mobile devices. This is primarily attributed to the fact that computation comprises more than 65% of the critical path during a page load. This illustrates the need to further optimize 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 focuses on improving the compilation and parsing time by introducing compile and parser cache. Recent studies [16] still report that the median page load time for a mobile website is about 14 seconds. Research [16] 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 [11] has been done to compare the page load times on mobile vs desktop, and

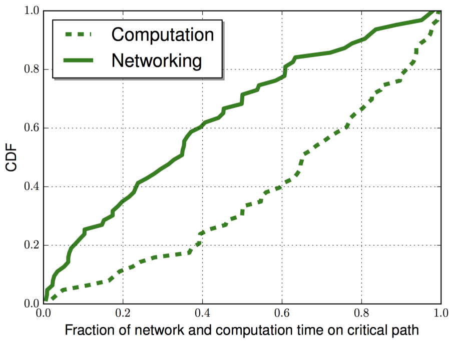

Figure 1: **Runtime information on mobile devices**

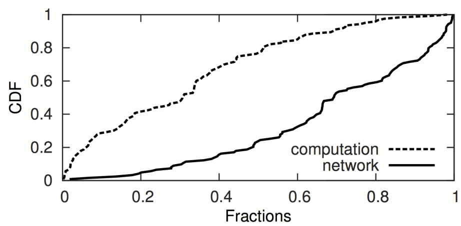

Figure 2: **Runtime information on desktops**

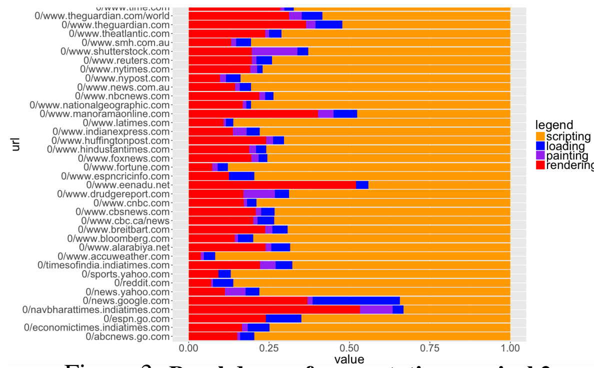

Figure 3: **Breakdown of computation on pixel 2**

recent results from 2016 claim that despite the increasing compute 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 significantly more time ,as compared to the other components of the total computation time. We break down computation into four categories: scripting, loading, paitnint, and rendering. and observe that scripting essentially takes more than 70% of the total computation time which is more than all the other categories combined (Figure 3). This makes it all the more important to do an in-depth analysis of the computation time to clearly understand where exactly this time is being spent.

3 Design

We propose a new technique to improve the page load time by reducing the javscript time. In order to do this, we intend to build a new caching framework for the modern web browsers, specifically Google Chrome. We choose to focus on Chrome because 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 simply the return value of the javascript functions. At other times it can be a modified DOM structure or just some intermediate result which will be further processed by other javascript functions, later down the execution timeline. The expiry of this javascript exectution cache is supposed to be same as the expire of the javascript source cache. There are a lot of caveats to this approach, and in our work we will explore all of these. The biggest challenge when developing a new caching framework is modifying the current browser's code in order to evaluate the efficacy of our caching framework. Since a lot of browsers already implement caching at the javascript runtime level, such as compiler and parser caches, a lot of this architecure can be borrowed for the execution cache as well. Another potential challenge will be the memory overhead. Most of the popular websites which spend about 70% of their time on javascript execution are running 1000s of javascript functions. Saving the output of all of these functions will add extra memory overhead to current browsers.

4 Progress

As a preliminary step in research, we first established a corpus of the top 75 news and sports websites on which we would run 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 a Google Pixel 2. We used Chrome v 61 to run all of our experiments on the mobile device. We leveraged chrome developer tools in order to capture the runtime traces, both for network and compute. We then analyzed 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.

We categorised computation time into four categorires, scripting, loading, rendering and painting. Scripting is the total time being spent on compiling, evaluating and executing javascript. Loading consists of parsing the html and css, which happens immediately after the payload for the network requests are received by the browser. Loading takes these payload objects and

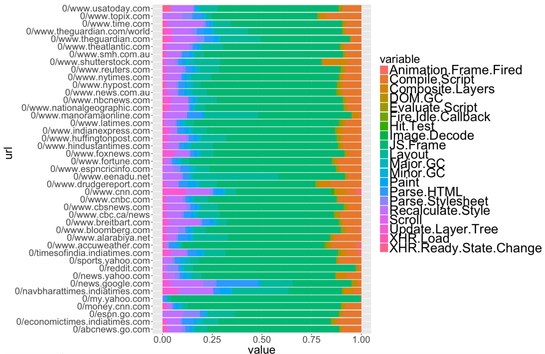

Figure 4: **Breakdown of computation into finer events on pixel 2**

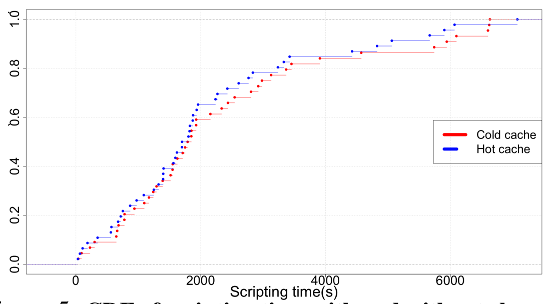

Figure 5: **CDF of scripting time with and without chrome's optimisations**

parses them before converting them to a DOM tree. Once the DOM tree is built, the rendering engine converts this DOM tree into a render tree, which contains the exact coordinates and the shape of each of the DOM node. This process comrpises the rendering time of the web page. Painting time is the time taken to process the render tree, and convert each pixel into a bitmap. Figure 3 shows the computation break down for these first level of categories for the Google Pixel 2. We further break down this time into the finer 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 in their 2017 dev summit, the Chrome team discussed 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 time with and without Chrome's optimizations to see the improvements. In order to do this, we captured the trace from Alexa's top 75 news and sports website once with a fresh cache, ie cold cache, and then subsequently with a hot cache which contains all of Chrome's optimizations, including its compiler and parser cache. As we can see from Figure 7, there has been a significant reduction in the overall compilation time, with about

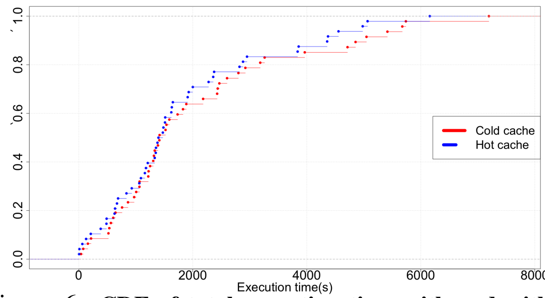

Figure 6: **CDF of total execution time with and without chrome's optimisations**

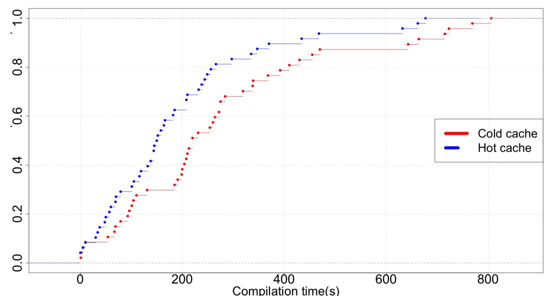

Figure 7: **CDF of compilation time with and without chrome's optimisations**

100ms reduction in median compile time. This is primarily due to the introduction of compiler and parser cache. The line corresponding to Cold cache refers to the fresh load of all the websites, whereas the line corresponding to the Hot cache refers to the subsequent load which makes use of Chrome's caching framework. This is also reflected partially in the overall scripting time as shown in Figure 5. Note that scripting time is the sum of compilation, execution and other minor javascript events in the execution pipeline like garbage collection. However the interesting thing to note, is that despite all these optimizations, we observe almost neglible improvement in the median execution time of the javascript, as shown in Figure 6. This serves as a huge motivation for the vast potential in improving the overall page load time, by optimizing the javascript execution time.

5 Related Work

Work on improving web performance has been ongoing 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 overall execution time. More recently, some work has been in improving the computatio time latency.

Erman et all [6] has shown that unlike desktop browsers, optimizations such as SPDY/HTTP2 does not improve PLT 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 [14] show that caching does not provide page load improvements for mobile browsers. A recent paper from Google in 2016 [20] showed how there is very little improvement to the overall page load time despite significant improvements in the cache hit rate.

Much of the research on explicitly improving mobile browser performance has seen mixed results. FLywheel [1] is Google's compression proxy that compresses web content to significanlty reduce the use of expensive cellular data. The authors note that whie Flywheel succeeds in reducing the data usage, its effect on page load time is more mixed; it helps the performance of certain pages and hurts the performance of others. Flexiweb [17] is built over Google's compression proxy to ensure that the proxy does not hurt page load times. However, FlexiWeb is not designed to explicitly improve page load performance. Wang et al [21] 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 that are orthogonal to the page load time metric. Parcel [19] uses a proxy approach to divide the page load process between the mobile device and the proxy. Because Parcel is a network appraoch, the evaluations are primarily focused on the reduction of network latency. Klotski [5] focuses on increasing the number of objects rendered in the first 5 seconds to improve the user quality of experience.

Other client side improvements reduce energy usage and computational delays using parallel browsers [9, 10] and improved hardware [24] By improving the parallelization for necessary page load tasks, such as rendering, these systems reduce energy usage and have a positive impact on page load times.

While there have been several recent efforts on improving mobile browser performance they have not been uniformly successful due to their various limitations.

References

[1] AGABABOV, V., BUETTNER, M., CHUDNOVSKY, V., COGAN, M., GREENSTEIN, B., MCDANIEL, S., PIATEK, M., SCOTT, C., WELSH, M., AND YIN, B. Flywheel: Google's data compression proxy for the mobile web. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2015)* (2015).

[2] BHATTI, N., BOUCH, A., AND KUCHINSKY, A. Integrating user-perceived quality into web server design. *Computer Networks 33*, 1 (2000), 1–16.

[3] BOUCH, A., KUCHINSKY, A., AND BHATTI, N. Quality is in the eye of the beholder: meeting users' requirements for internet quality of service. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (2000), ACM, pp. 297–304.

[4] BUTKIEWICZ, M., WANG, D., WU, Z., MADHYASTHA, H. V., AND SEKAR, V. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *NSDI* (2015), pp. 439–453.

[5] BUTKIEWICZ, M., WANG, D., WU, Z., MADHYASTHA, H. V., AND SEKAR, V. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (Oakland, CA, 2015), USENIX Association, pp. 439–453.

[6] ERMAN, J., GOPALKRISHNAN, V., RITTWIK, J., AND RAMAKRISHNAN, K. Towards a spdy'ier mobile web? In *CoNEXT* (2013), pp. 303–314.

[7] GUNDA, P. K., RAVINDRANATH, L., THEKKATH, C. A., YU, Y., AND ZHUANG, L. Nectar: Automatic management of data and computation in datacenters. In *OSDI* (2010), vol. 10, pp. 1–8.

[8] LÜTKEBOHLE, I. BWorld Robot Control Software. http://www.strangeloopnetworks.com/resources/infographics/webperformance-and-ecommerce/shopzilla-faster-pages-12-revenueincrease/, 2008. [Online; accessed 19-July-2008].

[9] MAI, H., TANG, S., KING, S. T., CASCAVAL, C., AND MONTESINOS, P. A case for parallelizing web pages. In *Proceedings of the 4th USENIX Conference on Hot Topics in Parallelism* (Berkeley, CA, USA, 2012), HotPar'12, USENIX Association, pp. 2–2.

[10] MEYEROVICH, L. A., AND BODIK, R. Fast and parallel webpage layout. In *Proceedings of the 19th International Conference on World Wide Web* (New York, NY, USA, 2010), WWW '10, ACM, pp. 711–720.

[11] NEJATI, J., AND BALASUBRAMANIAN, A. An in-depth study of mobile browser performance. In *WWW* (2016), pp. 1305–1315.

[12] NETRAVALI, R., GOYAL, A., MICKENS, J., AND BALAKRISHNAN, H. Polaris: Faster page loads using fine-grained dependency tracking. In *NSDI* (2016), pp. 123–136.

[13] NETRAVALI, R., SIVARAMAN, A., DAS, S., GOYAL, A., WINSTEIN, K., MICKENS, J., AND BALAKRISHNAN, H. Mahimahi: Accurate record-and-replay for http. In *USENIX Annual Technical Conference* (2015), pp. 417–429.

[14] QUIAN, F., QUAH, K. S., HUANG, J., ERMAN, J., GERBER, A., MAO, Z., SEN, S., AND SPATSHECK, O. Web caching on smartphones: ideal vs. reality. In *MobiSys* (2012), pp. 127–140.

[15] RUAMVIBOONSUK, V., NETRAVALI, R., ULUYOL, M., AND MADHYASTHA, H. V. Vroom: Accelerating the mobile web with server-aided dependency resolution. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), ACM, pp. 390–403.

[16] SHELLHAMMER, A. The need for mobile speed: How mobile latency impacts publisher revenue. https://www.doubleclickbygoogle.com/articles/mobile-speed-matters/, 2016. [Online; accessed September-2016].

[17] SINGH, S., MADHYASTHA, H. V., KRISHNAMURTHY, S. V., AND GOVINDAN, R. Flexiweb: Network-aware compaction for accelerating mobile web transfers. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking* (New York, NY, USA, 2015), MobiCom '15, ACM, pp. 604–616.

[18] SIVAKUMAR, A., PUZHAVAKATH NARAYANAN, S., GOPALAKRISHNAN, V., LEE, S., RAO, S., AND SEN, S. Parcel: Proxy assisted browsing in cellular networks for energy and latency reduction. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies* (2014), ACM, pp. 325–336.

[19] SIVAKUMAR, A., PUZHAVAKATH NARAYANAN, S., GOPALAKRISHNAN, V., LEE, S., RAO, S., AND SEN, S. Parcel: Proxy assisted browsing in cellular networks for energy and latency reduction. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies* (New York, NY, USA, 2014), CoNEXT '14, ACM, pp. 325–336.

[20] VESUNA, J., SCOTT, C., BUETTNER, M., PIATEK, M., KRISHNAMURTHY, A., AND SHENKER, S. Caching doesn't improve mobile web performance (much). In *USENIX Annual Technical Conference* (2016), pp. 159–165.

[21] WANG, X. S., BALASUBRAMANIAN, A., KRISHNAMURTHY, A., AND WETHERALL, D. Demysti-

fying page load performance with wprof. In *NSDI* (2013), pp. 473–485.

[22] WANG, X. S., BALASUBRAMANIAN, A., KRISHNAMURTHY, A., AND WETHERALL, D. How speedy is spdy? In *NSDI* (2014), pp. 387–399.

[23] WANG, X. S., KRISHNAMURTHY, A., AND WETHERALL, D. How much can we micro-cache web pages? In *Proceedings of the 2014 Conference on Internet Measurement Conference* (2014), ACM, pp. 249–256.

[24] ZHU, Y., AND REDDI, V. J. High-performance and energy-efficient mobile web browsing on big/little systems. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)* (Washington, DC, USA, 2013), HPCA '13, IEEE Computer Society, pp. 13–24.