Memoization of Browser Computations for a Faster Mobile Web

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 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 optimizations 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 the 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 recently 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 proportionate improvements 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 computation 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. Interestingly, 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

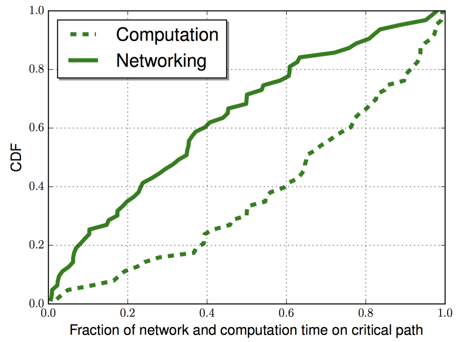

Figure 1: **Runtime information on mobile devices**

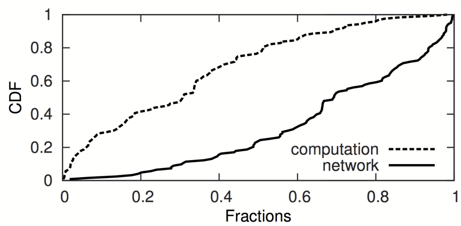

Figure 2: **Runtime information on desktops**

recent results from 2016 claim that despite the increasing compute resources in mobile devices, the computation time on mobile is significantly higher than their desktop counterparts. Our experiments on the most popular news and sports websites on the latest mobile hardware and the latest Chrome version reveal that despite these recent efforts, scripting still takes significantly more time than any other component. With our understanding of computation being the current bottleneck for high page load times, we conduct a set of experiments to evaluate the reasons for this exceedingly high computation time. In order to do this, we break down computation into four categories: scripting, loading, painting, and rendering. We observe that scripting 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. Using results from this study, we would eventually design our caching framework for the javascript execution output.

3 Design

We propose a new technique to improve the page load time by reducing the Javascript execution time, specifically the execution and script evaluation time. In order to do this, we intend to build a new caching framework for 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 of Javascript. Most of the time it is simply the return value of a Javascript function. At other times it can be a modified DOM structure or just an intermediate result which will be further processed as input to other Javascript functions. We further define Javascript execution output in Section 3.1.

The expiry of our implemented Javascript execution cache will be the same as the expiry of the Javascript source cache. The expiry is currently derived from the x-cache header field in the response, which determines how long the Javascript source will reside in the browser cache. There are many caveats to this approach, such as coming up with an optimal data structure to hold the Javascript execution cache and handling non determinism of the Javascript code. The biggest challenge when developing a new caching framework is modifying the massive code base of Google Chrome. However, Chrome already implements caching at the Javascript runtime level and we assume that much of this archicture can be borrowed for the execution cache as well. Another potential challenge will be the memory overhead. Most popular websites run thousands of Javascript functions and caching the output of all of these functions will add an extra memory overhead to current browsers.

3.1 Javascript Execution Output

The key idea for capturing a website's Javascript execution output is to capture the global changes made to the environment. This eliminates the need to capture any local computation, intermediate results computed, and any output that does not affect the global state of the browser.We define the global state of the browser to be represented by the window object, which is an instance of the open window/tab of the browser. This window object is supported by all the major browsers like Chrome, Firefox, Safari. If the execution of any javascript doesn't modify the window object in any way, then essentially that javascript has no affect to the global browser window state and therefore has no impact.

Modification of the global window object can be either from modifications to the current properties of the window object or the creation of new properties. Any global variable and function that is defined by the javascript becomes a new property of the window object.

3.2 Capturing JS Output

Our system for evaluating the changes in Javascript execution will observe the changes in outputs from the following granularities:

1. Website
2. Javascript file
3. Javascript function

We capture the entire window object state before and after the page has loaded to observe global changes to the website. We then perform a diff of these two window states to compare which properties have been modified during Javascript execution. This difference shows how much output must be cached and thus allows us to observe what the memory requirements for our caching framework will be.

We also capture the window object state after page loads across various time intervals to study how the global Javascript properties change over time. This data shows how much of the Javascript execution output can be cached and thus the impact caching can have on page load time.

To the best of our understanding, if 95% properties of the global window object remain unchanged across page loads, then we have a theoretical upper bound of a 95% reduction in Javascript execution time by employing our caching framework. We conducted a series of experiments to understand how much change is made to the window object when the page is loaded after three seconds (Figure 3), three hours (Figure 4) and three days (Figure 5). To our surprise, the change in the window object is extremely minimal, which leads us to expect a significant decrease in PLT with our caching framework.

However, Javascript files rarely contain single functions and thus to better understand the computation effects of a Javascript file, we evaluate the execution output at the function level.

To measure the effect of individual Javascript functions to the global state we create a call graph of the Javascript program during a page load. This dynamic call graph will be built during the page load. Each node in the graph represents an invoked function. Each node contains a signature for the function that was executed. This signature contains the function name, arguments, any global variables, and the return value. We use this signature to compare graphs across two different loads to establish how much of the execution can be cached.

We plan to cache Javascript execution at a function level. The results from the above two studies help us to define the upper bound of improved PLT with our caching framework. We have yet to implement this caching framework and evaluate the actual benefits of this approach.

4 Progress

As a preliminary step, we first established a corpus of the top 75 news and sports websites to cater to the most popular and compute intensive websites. These websites were gathered from the Alexa top website list. We ran all our experiments on a Google Pixel 2 with Chrome version 61. We leveraged chrome developer tools in order

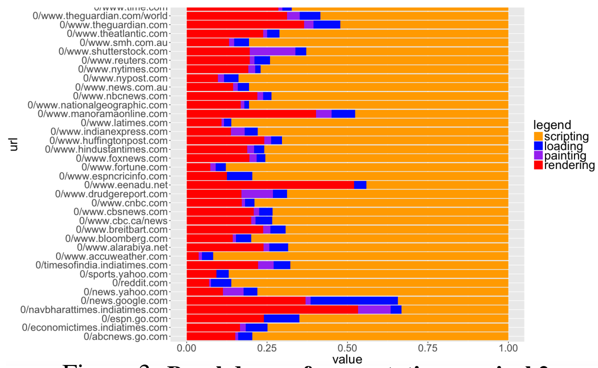

Figure 3: **Breakdown of computation on pixel 2**

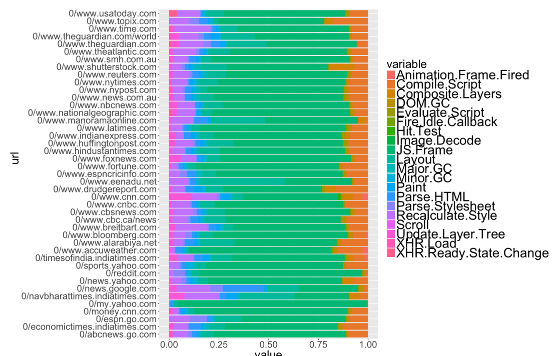

Figure 4: **Breakdown of computation into finer events on pixel 2**

to capture runtime traces for both networking and computation. We then analyzed these runtime traces to draw insight into the critical path of the website, the total computation time vs the total networking time, and most importantly the finer level breakdown of the computation time to understand the bottleneck of computation on mobile devices.

We categorized computation time into four categories: scripting, loading, rendering and painting. Scripting is the total time 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 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 nodes. This process comprises 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 6 shows the computation breakdown for these four categories on 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.

The results in Figure 7 show the promising impact a Javascript caching mechanism would have on the total

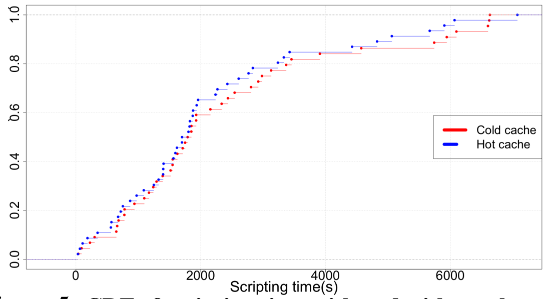

Figure 5: **CDF of scripting time with and without chrome's optimizations**

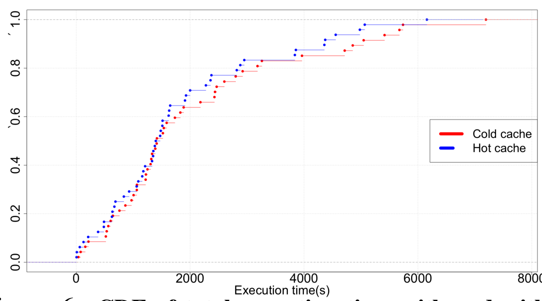

Figure 6: **CDF of total execution time with and without chrome's optimizations**

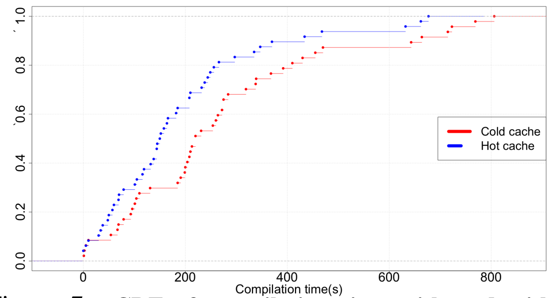

Figure 7: **CDF of compilation time with and without chrome's optimizations**

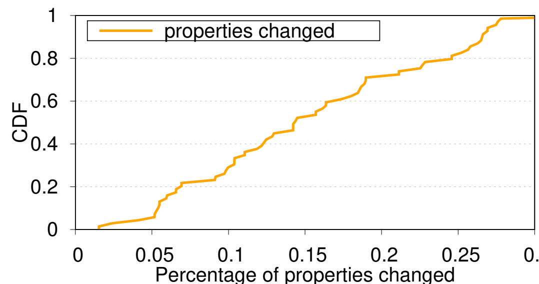

Figure 8: **Percentage of changed properties over 3 seconds**

page load time.

4.1 Current Chrome Optimizations

Recently in their 2017 dev summit, the Chrome team discussed the various optimization techniques they have developed to improve the total page load time. We did a comparison of the total page load time with and without Chrome's optimizations to study these improvements. We captured the trace from Alexa's top 75 news and sports website once with a fresh cache, i.e. cold cache, and then subsequently with a hot cache which contains all of Chrome's optimizations, including its compiler and parser cache. As seen in Figure 10, there has been a significant reduction in the overall compilation time, with about 100ms reduction in median compile time. This is primarily due to the introduction of the 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 8. Note that scripting time is the sum of compilation, execution and other minor javascript events in the execution pipeline such as garbage collection. However, the interesting thing to note is that despite all these optimizations, we observe almost negligible improvement in the median execution time of the Javascript, as shown in Figure 9. This serves as motivation for the vast potential a caching framework would have in improving the overall page load time.

After establishing the impact of a Javascript execution caching framework, we conducted experiments to understand Javascript computation at a finer granularity. We have explained the results from these experiments in section 3.1. We observed that most of the properties of the global window object remain unchanged. For a time difference of three seconds, we observe that only 2.5% of properties changed as shown in Figure 3. This is to be expected since little will change within three seconds of two subsequent web page loads. Surprisingly, even for a gap of three hours between two loads, only 3.5% of the properties changed as shown in Figure 4 and for a gap of three days, only 4.5% properties changed. These are the 95% percentile numbers, and therefore further motivate us to expect extremely high gains from a Javascript caching framework.

Currently, we are working on capturing the Javascript execution at the function level. In order to do this, we have built a web proxy which sits between the client browser and the news and sports websites. Every time a request is made by the client, the proxy intercepts the request, injects instrumentation code in the javascript files, and injects inline script tags inside the HTML files. This instrumented code is read by the Javascript debugger when the page is loaded, and the debugger then builds a call graph, with each node representing a function that was invoked. Once a graph is built, we will use a graph diffing algorithm to quantify how much of the call graph was modified across the two loads.

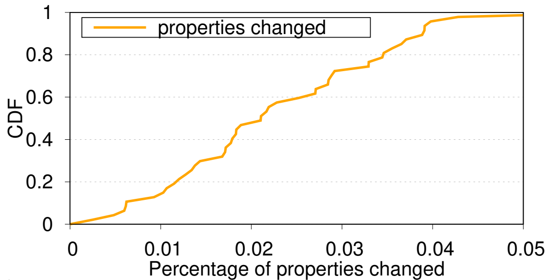

Figure 9: **Percentage of changed properties over 3 hours**

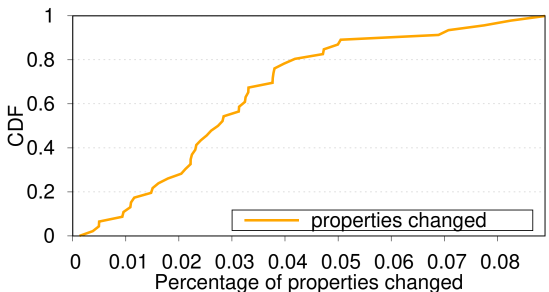

Figure 10: **Percentage of changed properties over 3 days**

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 re-modifying the source code of the web page itself to optimizing the network component of the overall execution time. More recently, some work has focused on improving the computation time latency.

Erman et all [6] has shown that unlike desktop browsers, optimizations such as SPDY/HTTP2 do 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. Similarly, 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 significantly reduce the use of expensive cellular data. The authors note that while 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 approach, 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.

6 Future work

Most of the project up till now has been to serve as a motivation for building a caching framework for the Javascript execution output. All of our experiments have shown the positive benefits of a caching framework. We also have established an upper bound on the posible decrease in the page load time in the best case scenario.

The actual implementation of the caching framework will be our next step. This is more like an engineering effort, which will determine the efficacy of our idea when we can fully evaluate the sytem. In order to do this, we will have to modify the production level source code of the Chrome browser.

Our understanding of the current caching framework for compiler and parser cache can be used as a reference for our Javascript caching framework.

7 Contribution

Our work had a fair distribution among the three co authors. This being the main research project of Ayush Goel, he was primarily responsible for setting up the testbed for the experiments and conducting them. All the code for the different analysis done for the page loads, like network analysis, trace analysis and capturing the window object state and running a simple diff algorithm is written by Ayush Goel. He has studied the current optimizations in place done by Chrome, and evaluated the improvement in the loading time with these optimizations enabled.

Matthew Furlong, the second co-author has contributed in finding the javascript libraries we currently use to capture the network and time line trace for each web page load, on top of which most of our other analysis code is written. He has contributed in analyzing data from our experiments by building CDFs of the results and comparing them with each other. He has also contributed assistance in writing the mid semester and final reports.

Hyunjong Lee, the third co-author has contributed in writing the mid semester and the final reports, generating figures and writing the content.

Most importantly, Matthew and Hyunjong were constantly involved in the critical discussion of the key ideas, and helped in establishing the next steps for our research project.

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.