Caching Doesn't Improve Mobile Web Performance (Much)

Jamshed Vesuna[†] Colin Scott[†] Michael Buettner[◇] Michael Piatek[◇]
Arvind Krishnamurthy[⋆] Scott Shenker[†‡]
[†]UC Berkeley [‡]ICSI [◇]Google [⋆]University of Washington

Abstract

A recent NSDI paper [1] reported that increasing the cache hit ratio for an HTTP proxy from 22% to 32% improved median page load time (PLT) for mobile clients by less than 2%. We argue that there are two main causes for this weak improvement: objects on the critical path are often not cached, and the limited computational power of mobile devices causes computational delays to comprise a large portion of the critical path.

Both of these factors were, in fact, outlined by a previous analysis of desktop web performance [2]. However, we (as the authors of the HTTP proxy [1]) did not properly understand the analysis and could have saved ourselves substantial engineering costs if we had. We therefore argue for the need to highlight this prior analysis, and extend the analysis to include mobile devices with slow CPUs, precise cache hit ratios, and a controlled reproduction of the HTTP proxy caching results [1]. In the extreme case of a perfect cache hit ratio, desktop page load times are improved notably by 34% compared to no caching, but mobile page load times only improve by 13% in the median case. We extract a back-of-envelope performance model from these results to help understand their underlying causes.

1 Introduction

Web caching is widely used to reduce network link utilization, decrease server load and data usage, improve reliability for origin web servers, and improve latency for end hosts. Here, we focus exclusively on web caching's effect on latency, as measured by web page load time.

Flywheel [1], Google's HTTP proxy for mobile devices, increased its overall cache hit ratio from 22% to 32%, yet observed only a 1–2% reduction in page load time in the median case. As the designers of Flywheel, we were initially surprised by this weak improvement. If we had been able to predict that caching would have such negligible effects, we could have saved ourselves substantial engineering costs.

In Sections 5 and 6 of their paper on measuring the critical paths of web page loads [2], Wang et al. seek to demonstrate use cases for their measurement tool. Two of their use cases—an analysis of varying CPU speeds, and an analysis of pages loaded with cold vs. warm vs. hot caches—in fact outline the likely root causes for Flywheel's result.

In this paper we seek to highlight Wang et al.'s analysis, as we suspect that we are not alone in holding the misconception that caching should improve latency. We also extend their analysis along several dimensions. We present a methodology for varying cache hit ratios at fine granularity, and measure caching's effects on web performance of both a mobile device and a desktop browser in a controlled and reproducible manner. In our controlled environment, we reproduce Flywheel's reported cache hit ratio increase for a set of 400 Alexa web pages [3] and find a comparable 1% decrease in PLT in the median case. In the extreme case of a perfect cache hit ratio, we find that desktop page load times are reduced notably by 34% compared to no caching, but mobile page load times are only reduced by 13% in the median case.

We develop a back-of-the-envelope performance model and fit its parameters to empirical observations to better understand the underlying causes. Our model indicates that CPU speed is the key resource bottleneck preventing mobile devices from benefiting significantly from web caching. The analysis by Wang et al. seems to further indicate that objects on the critical path are often not cached (or even cacheable).

Our analysis demonstrates that the generally favorable desktop latency improvements from caching do not carry over well to mobile clients. Content providers may want to think twice about expending resources on caching as a means for improving latency, especially as the volume of mobile traffic begins to overtake desktop traffic.

2 Background & Performance Model

Both browsers and mobile applications typically load content using the HTTP(S) protocol. When a user directs the browser (or application) to a new URL, the browser's Object Loader fetches the root HTML object, as depicted in Figure 1. The HTML Parser launches additional fetch requests for each linked resource within the HTML. In this way, the browser incrementally generates the DOM. As the page loads, the Rendering component paints the UI.

From the user's perspective, the performance of a website can be defined according to a number of different metrics [4,5]. Here, we focus on page load time, which is simple to measure and loosely standardized across browsers [6].

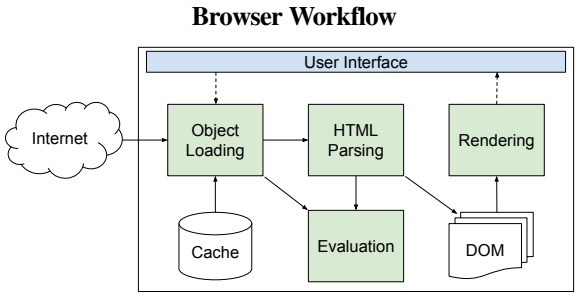

Figure 1: How a browser loads a web page. Reproduced from WProf [2].

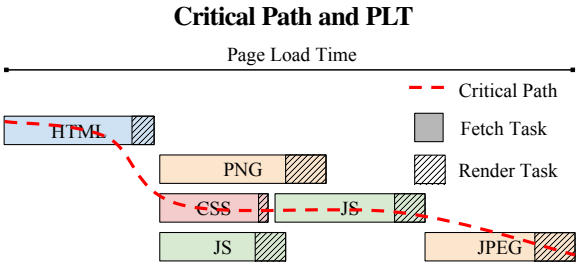

Figure 2: Page load time is determined from the critical path. Objects on the right are dependent on objects to their left, and objects at the same horizontal position are loaded concurrently.

Page Load Time. Roughly speaking, page load time (PLT) is the elapsed time from the moment a user requests a web page to the moment all resources on the page have been loaded [7]. We measure PLT by listening to the browser's JavaScript `onload` event, which fires in most browsers when all resources have been added to the DOM, and all images, scripts, links, and sub-frames have finished loading [6].

Critical Path. Web pages are comprised of many objects, such as images, JavaScript, CSS, and HTML. Each of these objects is handled by multiple (possibly concurrent) browser tasks: it must be fetched, parsed or evaluated, and rendered. We refer to the non-overlapping delays involved in parsing and rendering an object as its 'computational delay,' and refer to the fetch delay as its 'network delay.'

Critical path analysis is a method for analyzing the performance of parallel processes such as browsers. Certain load tasks are dependent on others and must wait until their predecessor tasks have completed. The critical path of a web page is the longest chain of dependent browser tasks such that reducing the length of any task not on the critical path will not change the page load time [8]. In Figure 2, the network and computational delay for the HTML, CSS, JS, and JPEG objects determine the PLT. If we were to decrease the delay for loading the PNG object, the critical path would remain the same, and therefore, the PLT would not change.

2.1 Performance Model

We can now develop an understanding of caching's effect on PLT with a simple performance model. First, consider the following terms:

– Let X denote a given cache hit ratio. We define cache hit ratio as the fraction of all objects in a web page that are served by a cache. Note that the maximum value of X is the fraction of cacheable items on the page, which may be less than 1.

– Let K denote the fraction of objects on the critical path that are cacheable.

– Let N denote the summation of network fetch delays for all objects on the critical path for a cold ($X=0$) page load.

– Let C denote the summation of computational delays for all objects on the critical path for a cold ($X=0$) page load.

– Let $f(X)$ denote the overlap between computational delay and network delay on the critical path. Normally, dependent objects on the critical path should not overlap. There are, however, some cases where the browser can begin concurrently loading an object when its predecessor is only partially loaded. For most purposes, we can ignore $f(X)$.

For simplicity, let us assume that (i) the critical path does not change as we vary the cache hit ratio, (ii) the probability of an object being in cache is uniform across all cacheable objects, and (iii) cached items incur zero network delay. The probability of an object on the critical path incurring a network delay is then:

$$1-Pr(\text{cacheable})\cdot Pr(\text{cache hit}\,|\,\text{cacheable})$$

The expected value of the PLT for a given X is therefore:

$$E_{PLT}[X]=C+(1-K\cdot X)\cdot N-f(X)$$

2.2 Fitting the Model

Sections 5 and 6 of the WProf paper [2] contain empirical measurements of critical paths that allow us to gain a rough understanding of the values of N, C, and K in our model above.

Fitting N and C. In Figure 3, we reproduce WProf's "what-if" analysis (Figure 13 from WProf) for torchbrowser.com. This experiment investigates the performance impact of varying network and computation speeds. We first multiply the computational or network delays for all objects in a web page by a fixed constant. Then, we recompute the page's critical path (based on task dependencies captured by WProf), and extract a predicted PLT. The comp=1 line represents the (2 GHz) desktop CPU that loaded the original page, while comp=0 represents an infinitely fast CPU, comp=1/2 represents a CPU that is twice as fast, and comp=2 (not present in WProf's analysis) represents a CPU of half the speed.

For the infinitely fast CPU (comp=0) we see that its normalized PLT with an unchanged network speed (ratio of network time = 1) should be ~0.8. As we improve network delays for this CPU, we should see a theoretically infinite speedup (tending towards a PLT of 0). Conversely, for the slowest CPU (comp=2), the normalized PLT for an infinitely fast network (ratio of network time = 0) is ~0.4. For this hypothetical CPU (assuming $f(\cdot)$ is close to 0), we can

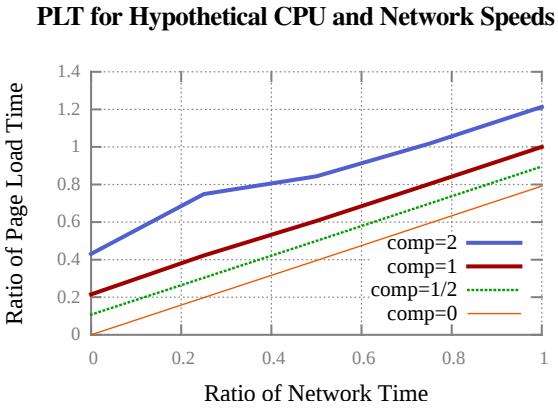

Figure 3: Predicted PLT for torchbrowser.com when hypothetical computation and network speeds are varied.

estimate that the fraction of the critical path that consists of computational delays is ~0.4, while the fraction of the critical path that consists of network delays is ~0.6.

The key takeaway from this analysis is that, as we decrease the speed of the CPU, the ratio of $C:N$ continues to increase. For example, our analysis suggests that a typical mobile device with a ~1 GHz CPU [9] has a $C:N$ ratio of ~2/3 for websites similar to torchbrowser.com. This makes intuitive sense: slower CPUs would cause computational delays to make up larger fractions of the critical path compared to faster CPUs.

Fitting K. To generate Figure 11b in WProf [2], the authors measured the fraction of objects that were in cache immediately after loading pages with a cold cache. Although 65% of all objects were cached, only 20% of all objects on critical paths were cached, giving us an estimate of $K = 0.2$.

Implications. The analysis by WProf, together with our model, give us a rough understanding of the performance effects from caching. We return to our model in §4, where we seek to gain a deeper empirical understanding.

3 Experimental Apparatus

In order to gain a deeper understanding of the performance dynamics outlined in §2.1 and §2.2, we need to experimentally evaluate questions about how caching affects PLT in a controlled environment. Here, we develop our methodology.

Our experimental apparatus (publicly available at [10]) makes use of Telemetry [11] and Web Page Replay (WPR) [12] to measure the effects of parameterized levels of network delays. Both Telemetry and WPR are part of Chromium [13], the open source components of Google Chrome. Here, we use the term 'browser' interchangeably with Google Chrome.

Web Page Replay. WPR acts as a local DNS and HTTP(S) proxy cache (depicted in Figure 4a). In record mode, WPR forwards HTTP(S) requests to the Internet, and records all requests and responses that it observes, as well as metadata such as observed network delays. From this, Web

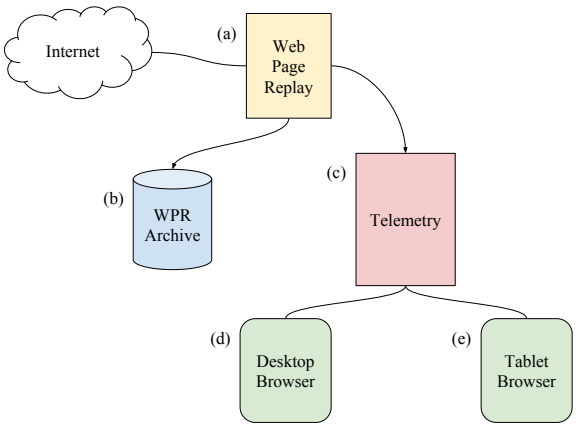

Figure 4: Logical units of our apparatus.

Page Replay builds a WPR archive (depicted in Figure 4b), complete with all HTTP(S) requests, responses, headers, data, and network delays.

In replay mode, the WPR proxy responds to HTTP(S) requests with the responses saved in the archive, or a 404 error response if the corresponding response is not saved in the archive. We configure WPR to send the matching response and data only after sleeping for the time duration originally observed as the network delay between the origin and the WPR proxy (while it was in record mode). Here, WPR is emulating an edge cache rather than a browser cache.

Telemetry. Telemetry (Figure 4c) is a browser performance testing framework, which orchestrates the behavior of the browser and WPR. We use Telemetry to control one of two browsers. The first is a desktop version of Google Chrome running within a virtual machine (specifications in 3.2). The second is a mobile version of Google Chrome running on a USB-tethered mobile device. We use Telemetry to load requested URLs in the browser (as if the user is entering URLs into the omnibox) and passively measuring PLT.

3.1 Workflow

Before each experiment, we clear the browser cache to ensure consistency across trials. For each device (desktop and mobile), we execute the following steps for each URL:

First, we record the live web page from the Internet using Telemetry to instruct the browser to fetch the given URL. The WPR proxy receives this HTTP(S) request, forwards it to the Internet, and passively inspects and records the two-way traffic as noted in §3. We store this data as a WPR archive.

Next, we determine the page load time of the web page with a cold cache. With WPR in replay mode, we load the URL four times (cf. §3.3) and take the minimum page load time as our PLT value, to account for variance.

Now, we emulate a "perfect," fully populated cache. First, we copy the original WPR archive into a new WPR archive.

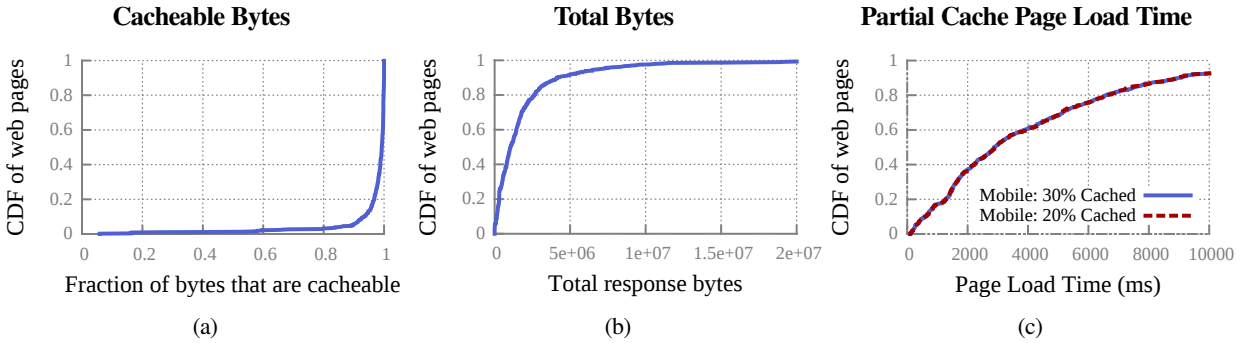

Figure 5: (5a): The majority of web pages are composed mostly of cacheable bytes. (5b): While 95% of web pages are under 6.8 MB, the median web page size is less than 1.2 MB. (5c): Increasing the cache hit ratio from 20% to 30% had negligible effects on mobile PLT.

For each cacheable response in this new WPR archive, we set its network delay to 0 (of course, a "real" cache response time of 0 is not possible, but we set this as an absolute lower bound). Non-cacheable items, as indicated by HTTP headers, retain their initial network delays. We store this modified archive alongside the original (Figure 4b).

Lastly, we record the page load time again, however this time using the modified WPR archive. We determine the PLT in the same way as the original. We then compare the page load times of the unmodified replay executions to that of the modified "perfect cache" executions.

Partial Caching Methodology. To confirm Flywheel's findings, we created two additional sets of partially cached WPR archives: one that caches a randomly chosen set of 30% of all cacheable resources (regardless of byte size), and another that caches 20%. We ensure that the cached items in the 20% WPR archive are a strict subset of the cached items in the 30% WPR archive for consistency.

3.2 Specifications

Each web page was originally fetched over UC Berkeley's LAN, which approximates 250 Mbps down and 230 Mbps up. Our mobile device is a Galaxy Tab 4 with a 1.2 GHz quad-core processor and 1.5 GB on board RAM running Android 4.4, KitKat. Desktop results were performed in a virtual machine with a 3.2 GHz quad-core processor and 6 GB RAM.

3.3 Known Limitations

We identify the following limitations of our apparatus, and discuss the reasoning behind our choice of tools:

Page Load Time as a Metric. When determining web page performance, we chose to focus on page load time rather than SpeedIndex [5] or above-the-fold time [4]. Although they are arguably preferable metrics (as they do a better job of capturing the user's perspective), these metrics are significantly more difficult to measure.

WPR Measurement Accuracy. The PLT measurements taken by WPR are not necessarily consistent with PLTs observed on live web pages, nor are they necessarily consistent across multiple runs of WPR. First, although WPR attempts to mitigate non-determinism in JavaScript execution (by injecting a script into each web page that interposes on non-deterministic calls such as `getTime`), JavaScript may nonetheless exhibit non-determinism across different loads. Second, the mechanism WPR uses to emulate the original RTTs observed during record mode (sleeping a fixed number of milliseconds) may not perfectly match the behavior of the original page load. We try to mitigate these artifacts by loading each web page four times and taking the minimum PLT.[1]

4 Results

Here, we demonstrate empirical performance results we have found with our apparatus. We also attempt to highlight the underlying effects that determine our results.

4.1 Workload Characteristics

We first note several key characteristics of our data corpus:

Data Set. We selected a random subset of 400 of the Alexa top 2000 URLs [3] and loaded their root URL ('/').

Fraction of Cacheable Bytes. Over 90% of web pages in our workload have more than 90% of their total bytes marked as cacheable, as shown in Figure 5a.

Total Bytes. Figure 5b shows the spread of web page sizes in our data set. While 95% of web pages are under 6.8 MB, the median web page size is less than 1.2 MB.

Initial Network Delays. Across all requests/response pairs, the median delay between sending the request and receiving the first response byte was 50ms, with a mean of 151.17ms and standard deviation of 403.77ms.

[1] We observed that beyond four loads per web page, the minimum PLT value did not decrease significantly.

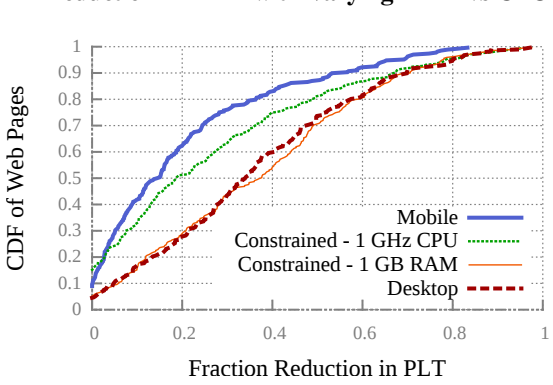

Figure 6: Reduction in page load time due to (perfect) caching is significantly less on mobile devices than desktop. Further, CPU, not RAM, is the primary resource that differentiates mobile devices from their desktop counterparts.

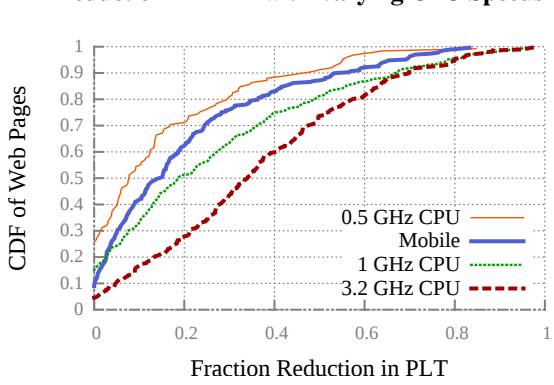

Figure 7: Slower CPU speeds cause increasingly diminished benefits from (perfect) caching.

User Agent. Many web pages are now optimized for mobile devices. Web servers inspect the user agents (UA) of incoming HTTP(S) requests to deliver customized content to the client depending on their device size and computational resources. We ran all of our experiments twice: once where the browsers (both desktop and mobile) advertised a mobile UA, and once where the browsers advertised a desktop UA. We found that the differences in the results were comparable. Here, we show only the desktop UA results to make our graphs more readable.

4.2 Performance Results

As we saw in Figure 5a, 90% of pages are composed of >90% cacheable bytes. If network delays were dominant and the fraction of cacheable objects on the critical path were moderately high, one would conclude that PLT would become negligible with a perfect cache. As our model predicts however, this is not the observed outcome.

Caching Doesn't Significantly Reduce Mobile PLT. We reproduced Flywheel's result in our controlled environment by emulating cache hit ratios of 20% and 30%. As shown in Figure 5c, we found that increasing the hit ratio by 10 percentage points had negligible effects on mobile page load time. Consistent with the reported Flywheel result, we observed only a 1% reduction in PLT in the median case. With a *perfect* cache (Figure 6), our mobile device gains only a 13% PLT reduction in the median case, while its desktop counterpart sees a PLT reduction of 34%.

Limited RAM Does Not Affect Computational Delays. It is possible that either limited RAM or limited CPU would increase computational delays on the critical path. Here, we seek to isolate which of these resources plays a larger role.

A typical mobile device in the global market today has a 1 GHz processor and 1 GB of RAM [9]. We emulate these conditions and isolate computational resources with virtual machines that were constrained by different resources (using VirtualBox to either set a limit on memory usage or emulate a slower CPU clock speed). With 'Mobile' and (unconstrained) 'Desktop' as baselines, Figure 6 presents a stark contrast between these resource constraints: our CPU constrained VM ('Constrained - 1 GHz CPU') behaves very similarly to our tablet ('Mobile'), while our RAM constrained VM ('Constrained - 1 GB RAM') is more closely aligned with the 'Desktop' results. From this we conclude that CPU, not RAM, plays the larger role in determining the performance improvements from caching.

Slow CPU Speeds Increase Computational Delays. Now that we have isolated CPU as the bottleneck resource, we seek to measure the extent of its impact. In terms of our model, we already know that slower CPUs should incur higher computational delays, but here we seek to understand the empirically observed ratios of $C : N$ (as opposed to the hypothetical, predicted ratios in Figure 3). In Figure 7, we observe that, as predicted, as we throttle CPU constraints, perfect caching has noticeably smaller effects on PLT.

The Marginal Benefits of Caching Sharply Decrease. Figure 8 illustrates that for each 10 percentage point increase in cache hit ratio, there is only a 1 percentage point decrease in mobile page load time. That is, there is a sharply diminishing marginal performance gain for every additional byte that is cached. This experimental evidence supports our model: although we do not directly measure the critical path (since WProf is not currently available for mobile browsers), it appears that the fraction of cacheable bytes on the critical path (K) is significantly smaller than the fraction of cacheable byes *not* on the critical path.

4.3 Data Validation

We made several efforts to sanity check our results [14]. To mitigate non-determinism, we compared the status codes

[2]The fraction reduction in PLT for a web page is defined as (Original PLT - PLT with a perfect cache) / (Original PLT).

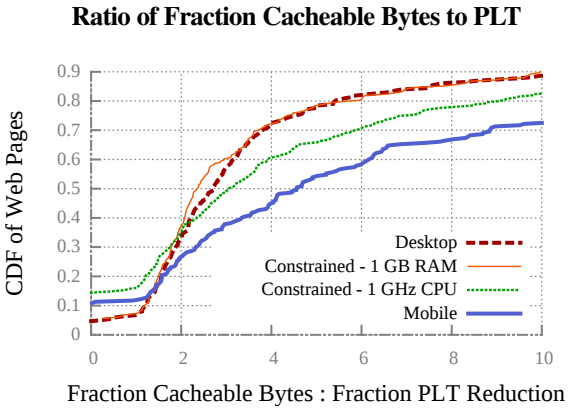

Figure 8: As the percentage of cacheable bytes in a web page increases, the reduction in page load time due to caching increases. However, for each additional percentage cached, there is less than a percentage reduction in page load time.

of all objects loaded in the browser from both original and perfect/partial cache benchmarks. We filtered out about 9% of web pages in our 400 URL data set in cases where there were a high number of 404 error codes due to non-deterministic requests without responses in the WPR archive. The figures we present show only these 91% of web pages that passed our non-determinism filter.

As the ratio of cached to non-cached bytes increases in a web page, we expect page load time to be less than or equal to that of its non-cached counterpart. As seen in Figure 8, there is a positive correlation between the fraction of cached bytes and the reduction in PLT, albeit asymptotic. However, due to variance in PLT (discussed in §3.3), we see in Figure 6 that ~10% of web pages perform worse when cached, as indicated by the data points to the left of `X = 0`.

5 Related Work

Several papers have analyzed web performance, caching, and the relationship between CPU speeds and PLT.

WProf. Wang et al. [2] is the closest research to ours. As we discussed in §2, the experiments WProf ran for their Figure 11 show that objects on the critical path are often not cached; and the experiments they ran for their Figure 13 implies that decreasing CPU speed causes computational delays to comprise a larger fraction of the critical path.

We extend their research along several dimensions. We develop a model that allows us to predict PLT for a given cache hit ratio. We show that limited RAM does not increase computational delays, though slow CPUs do. We also empirically measure (rather than statically compute, as WProf does) PLTs using a tablet device, and using CPU-constrained virtual machines, over a larger data set (400 URLs, vs. ~50 URLs). Lastly, we extend WProf's cacheability analysis to show that the marginal returns from caching sharply diminish.

Concurrently with our work, Nejati et al. ported WProf to mobile browsers [15]. Although they do not consider caching, their tool would be invaluable for deepening our analysis.

Web Performance. Related studies [16, 17] focus on evaluating and optimizing web performance for desktops. Many techniques such as altering content, data compression, proxy services, and CDNs have been exploited to reduce latency for users. These studies focus on high performing end devices such as desktops. We additionally analyzed and compared web performance on a mobile device.

Zhen Wang et al. [18, 19] have determined that the largest delay factor in desktop web page loading is object rendering in the browser. They went further to show that CPU constraints are the lead cause of slow resource loading. With a large data set, we bolster their claim that CPU constraints are the critical factor in determining page load time. We also demonstrate that web caching has diminishing benefits due to the limited CPU speeds of mobile devices.

We are not the first to focus on web performance for mobile devices [18, 19]. Our main contribution is developing a performance model for pinpointing the key differences between desktop and mobile.

Web Caching. Other literature [20–23] has focused on the benefits of web caching, specifically the reduction of latency for desktops [24–28]. While these papers make note of the several benefits of caching, they do not focus on highlighting caching's effects on (CPU-constrained) mobile devices.

Proposed Changes to the Web. There are many papers [29–37] that propose changes to the web that would improve web latency with better caching schemes. It is possible that under their proposed changes, caching would have more of a benefit for mobile latency. In this paper, we focus only on today's existing infrastructure.

6 Conclusion

Motivated by our initial surprise at Flywheel's weak performance improvements from smarter caching, we sought to highlight and extend the analysis done by Wang et al. [2], which indicates two reasons caching should not significantly improve page load time for mobile devices: slow CPU speeds, and sparsity of cached items on the critical path. To make effective use of caching, content providers should pay careful attention to whether cached objects are on the critical path.

Going forward, mobile devices are becoming increasingly powerful, and the bottleneck resources will shift. We hope that the model we have developed here will help content providers and network designers make informed decisions about the performance effects of caching for the mobile web of the future.

Acknowledgments. We thank the anonymous reviewers for their feedback, and especially our shepherd Dan Tsafrir for helping us develop our performance model. This research was supported by an NSF Graduate Research Fellowship.

References

[1] V. Agababov, M. Buettner, V. Chudnovsky, M. Cogan, B. Greenstein, S. McDaniel, M. Piatek, C. Scott, M. Welsh, and B. Yin. Flywheel: Google's Data Compression Proxy for the Mobile Web. NSDI '15, 2015.

[2] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. Demystifying Page Load Performance with WProf. NSDI '13, 2013.

[3] Alexa Internet Inc. Alexa Top 500 Global Sites. http://www.alexa.com/topsites, 2015. Accessed: 2015-9-12.

[4] Pat Meenan Jake Brutlag, Zoe Abrams. Above the Fold Time: Measuring Web Page Performance Visually. O'Reilly Media, Inc., 2011. Accessed: 2016-1-19.

[5] Google. Speed Index. https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index.

[6] W3C. Navigation Timing Level 2 Spec. http://w3c.github.io/navigation-timing/.

[7] Google Developers. PageSpeed Insights, 2015. Accessed: 2016-1-11.

[8] Vivek Sarkar. Partitioning and scheduling parallel programs for execution on multiprocessors. Technical report, Stanford Univ., CA (USA), 1987.

[9] Statista. Global market share held by leading smartphone vendors from 4th quarter 2009 to 3rd quarter 2015, 2015. Accessed: 2016-1-26.

[10] Jamshed Vesuna. Telemetry, Web Page Replay Experimental Apparatus. https://github.com/JamshedVesuna/telemetry.

[11] Google. Telemetry. https://catapult.gsrc.io/telemetry.

[12] Google. Web Page Replay. https://github.com/chromium/web-page-replay.

[13] Google Chromium. Google. https://www.chromium.org/.

[14] Jamshed Vesuna. Sanity Checks. https://github.com/colin-scott/page_load_time/tree/master/telemetry/sanity_checks.

[15] Javad Nejati and Aruna Balasubramanian. An In-Depth Study of Mobile Browser Performance. WWW, 2016.

[16] Steve Souders. High-performance web sites. *Communications of the ACM*, 2008.

[17] Patrick Killelea. *Web Performance Tuning: Speeding up the Web*. " O'Reilly Media, Inc.", 2002.

[18] Zhen Wang, Felix Xiaozhu Lin, Lin Zhong, and Mansoor Chishtie. Why are Web Browsers Slow on Smartphones? In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*. ACM, 2011.

[19] Zhen Wang, Felix Xiaozhu Lin, Lin Zhong, and Mansoor Chishtie. How Far Can Client-only Solutions Go for Mobile Browser Speed? In *Proceedings of the 21st international conference on World Wide Web*. ACM, 2012.

[20] Bernhard Ager, Fabian Schneider, Juhoon Kim, and Anja Feldmann. Revisiting Cacheability in Times of User Generated Content. In *INFOCOM IEEE Conference on Computer Communications Workshops, 2010*. IEEE, 2010.

[21] Jia Wang. A Survey of Web Caching Schemes for the Internet. *ACM SIGCOMM Computer Communication Review*, 1999.

[22] Lothar Braun, Alexander Klein, Georg Carle, Helmut Reiser, and Jochen Eisl. Analyzing Caching Benefits for YouTube Traffic in Edge Networks A Measurement-based Evaluation. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012.

[23] Pei Cao and Sandy Irani. Cost-Aware WWW Proxy Caching Algorithms. In *Usenix symposium on internet technologies and systems*, 1997.

[24] David A Patterson. Latency Lags Bandwith. *Communications of the ACM*, 2004.

[25] Armando Fox and Eric A Brewer. Reducing WWW Latency and Bandwidth Requirements by Real-time Distillation. *Computer Networks and ISDN Systems*, 28, 1996.

[26] Kun-Lung Wu and S Yu Philip. Latency-sensitive Hashing for Collaborative Web Caching. *Computer Networks*, 2000.

[27] Pablo Rodriguez, Keith W Ross, and Ernst W Biersack. Improving the WWW: Caching or Multicast? *Computer Networks and ISDN Systems*, 1998.

[28] Swaminathan Sivasubramanian, Guillaume Pierre, Maarten van Steen, and Gustavo Alonso. Analysis of Caching and Replication Strategies for Web Applications. *Internet Computing, IEEE*, 2007.

[29] Leo A Meyerovich and Rastislav Bodik. Fast and Parallel Webpage Layout. In *Proceedings of the 19th international conference on World wide web*. ACM, 2010.

[30] Jeffrey Erman, Alexandre Gerber, Mohammad T Hajiaghayi, Dan Pei, and Oliver Spatscheck. Network-aware Forward Caching. In *Proceedings of the 18th International Conference on World Wide Web*. ACM, 2009.

[31] Kaimin Zhang, Lu Wang, Aimin Pan, and Bin Benjamin Zhu. Smart Caching for Web Browsers. In *Proceedings of the 19th international conference on World wide web*. ACM, 2010.

[32] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z Broder. Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol. In *ACM SIGCOMM Computer Communication Review*, volume 28. ACM, 1998.

[33] John Dilley and Martin Arlitt. Improving Proxy Cache Performance: Analysis of Three Replacement Policies. *IEEE Internet Computing*, 1999.

[34] Guohong Cao. A Scalable Low-latency Cache Invalidation Strategy for Mobile Environments. *Knowledge and Data Engineering, IEEE Transactions on*, 2003.

[35] Guohong Cao. Proactive Power-aware Cache Management for Mobile Computing Systems. *Computers, IEEE Transactions on*, 2002.

[36] Sunho Lim, Wang-Chien Lee, Guohong Cao, and Chita R Das. A Novel Caching Scheme for Improving Internet-based Mobile Ad Hoc Networks Performance. *Ad Hoc Networks*, 4, 2006.

[37] Chanda Dharap. Semantics-based Caching Policy to Minimize Latency, 1999. US Patent App. 09/374,694.