Demystifying Hardware Bottlenecks in Mobile Web Quality of Experience

Mallesham Dasari
Stony Brook University
mdasari@cs.stonybrook.edu

Conor Kelton
Stony Brook University
ckelton@cs.stonybrook.edu

Javad Nejati
Stony Brook University
jnejati@cs.stonybrook.edu

Aruna Balasubramanian
Stony Brook University
arunab@cs.stonybrook.edu

Samir R. Das
Stony Brook University
samir@cs.stonybrook.edu

ABSTRACT

Mobile web page load time depends on three key factors: (1) the complexity of the Webpage, (2) the underlying network conditions, and (3) the processing capability of the device. While there are several works focusing on the Web complexity and the network, there is a little work in understanding the hardware bottlenecks in the page load process. In this poster, we analyze the effect of hardware bottlenecks of Web pages. We also analyze the effect of GPU offloading, a commonly used solution to speed up Web page loads.

CCS CONCEPTS

• **Hardware** → **Networking hardware**; Power and energy; • **Networks** → *World Wide Web (network structure)*;

KEYWORDS

Web QoE, CPU-GPU, Page Load Time

ACM Reference format:
Mallesham Dasari, Conor Kelton, Javad Nejati, Aruna Balasubramanian, and Samir R. Das. 2017. Demystifying Hardware Bottlenecks in Mobile Web Quality of Experience. In *Proceedings of SIGCOMM Posters and Demos '17, Los Angeles, CA, USA, August 22–24, 2017,* 3 pages.
https://doi.org/10.1145/3123878.3131980

1 INTRODUCTION

Typically, mobile web users experience poorer page load performance compared to desktop users. Our goal is to determine the root cause for this poor performance. Figure 1 shows factors influencing the page load at different layers : Application layer (website complexity, encryption overheads), OS/HW layer (memory and processing bottlenecks) and Network sub layer (network wide parameters).

With the increasing speeds of Wi-Fi and Cellular connection, mobile page loads are no longer bottlenecked by the network alone. Compute tasks such as parsing (e.g HTML, Javscript), scripting (CSS, JS), layout, and painting introduces huge computation bottleneck.

SIGCOMM Posters and Demos '17, August 22–24, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5057-0/17/08.
https://doi.org/10.1145/3123878.3131980

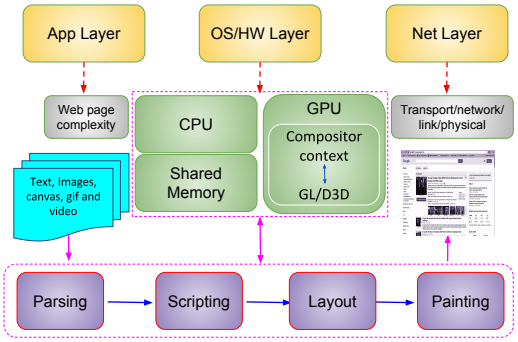
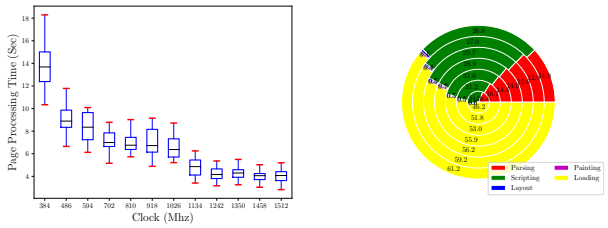


Figure 1: Architecture of Page Load Process

This problem is especially important for low-end devices. Over 68% [2] of mobile users from developing regions (specifically India and African countries) use phones that are not powerful. This motivates us to explore hardware bottlenecks in the page load process as a reason for poor page performance. Interestingly, we find that hardware bottlenecks exist even for higher-end mobile devices, with the bottlenecks getting worse as the mobile devices get cheaper.

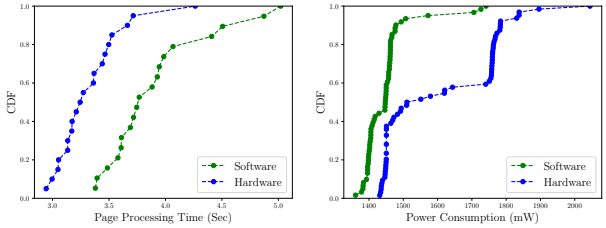
Prior work has shown that computation is a bottlebeck in mobile devices [4, 7]. The WProf-M [4] work shows that while desktop browsers are limited mostly by network, mobile browsers are bottlenecked by compute. The Webcore [7] work optimizes the mobile hardware architecture to improve PLT and minimize energy consumption. Many mobile browsers (Chrome, Firefox and IE) have introduced GPU accelerators to speed up web page loads. Given these related works, our contributions include:

- Fine-grained measurements to study the effect of the underlying hardware resources on mobile page load performance.
- Dissecting the page load time into critical stages (loading, parsing, scripting, layout and painting) and finding critical blockage points in terms of compute.
- Exploring the use of GPU accelerated compositing (hardware rendering) over software rendering.



(a) (b)

Figure 2: (a): PPT versus Clock Frequency, (b): Dissection of PLT into Loading, Parsing, Scripting, Layout, Painting



(a) (b)

Figure 3: (a): PPT versus Hw-Sw Rendering, (b): Power Consumption versus Hw-Sw Rendering

2 EFFECT OF CLOCK FREQUENCY AND MEMORY

We first study the effect of the underlying hardware on page load time (PLT). To this end, we exclude loading time and measure only Page Processing Time (PPT) at the client, with respect to different frequency governors and processor clock rate to emulate different mobile devices.

SetUp: Our experiments were performed on a Google Nexus5 Android phone running the Chrome browser. We load the top 20 most popular pages in Alexa for our experiments. We use chrome developers tools [3] to trace browser events and calculate start and end times for all events: parsing, scripting, layout and painting. We collect hardware resource consumption statistics using Snapdragon Profiler from Qualcomm [5]. We change CPU clock frequency to emulate different low-end mobile devices clock using android debug bridge tool interface on Linux. Typically, android governors control frequency to dynamically adapt to balance performance and power consumption. We experiment with different frequency governors available on the Nexus5 phone. To minimize variances due to Internet delays, we emulate static network conditions using the Linux Traffic Control tool [1]. Memory is constrained on smartphones, having 56.7% of the devices less than 1GB of RAM [2]. Hence, to study the impact of memory, we change memory availability by creating RAM disks (in steps of 256MB) from available memory and assign RAM disk to memory intensive workloads to occupy completely.

Results: We observe a median PPT difference of 10 seconds between devices with low CPU frequency (384Mhz) and high CPU frequency (1512Mhz) (Figure 2(a)). This is averaged for four available governors on Nexus5 (powersave, performance, interactive and on-demand) over 50 runs. We also find that even at a high frequency, there is a median 3 second processing delay, which increases to 14 seconds when the CPU frequency is low.

Next, we analyze various components that make up the PLT including parsing, scripting, rendering, painting and compare it with the time taken for loading objects (networking). We perform this study as a function of clock rate. We use the WProf tool [6] to obtain fine-grained component level breakdown of the PLT on the critical path.

Figure 2(b) shows that, as the clock frequency increases, the fraction of time spent on network decreases. When moving from the highest to the lowest clock frequency, the fraction of network loading decreases by a median of 18.6%. For lower frequencies, scripting (36.1%) and parsing (19.6%) make up a large amount of the critical path, while layout and painting contribute negligibly at all frequencies(0.5% each). Furthermore, with better hardware, the fraction of time spent on scripting and parsing decreases by a median of 7.8% each.

We find that memory availability is also a critical issue. When memory is increased from 512MB to 2GB, the PPT measure reduced by 8 seconds in the median case, which is a reduction of 12.5%.

3 EFFECT OF GPU OFFLOADING

Next we study the effect of offloading browser computation to the GPU. Due to the highly parallel nature of web content, modern browsers are leveraging GPUs to improve page load performance by accelerating composting layers on GPU. Compositing pages on the GPU especially helps Webpages that are image- and video-heavy. However, using the power-hungry GPU is a problem for battery life particularly when using high performance CPU and GPU governors. We study this trade-off.

We measured page processing time with respect to both software compositing and hardware-accelerated compositing using GPUs. We set the CPU clock to 1512Mhz and the GPU clock to 400Mhz with interactive frequency governor. Figure 3(a) shows that offloading compositing to the GPU reduces the PPT by a median value of 0.5 seconds.

We measure power consumption (using Snapdragon Profiler) during both software and hardware rendering. Figure 3(b) shows that by using the GPU for rendering, the power consumption increases by 22% for 90% of the time. In other words, even though the GPU offloading may improve performance significantly, it is also more power hungry. Our experiments are limited to Chrome browser.

4 ONGOING AND FUTURE WORK

We are currently working on understanding the effect of hardware resources on the mobile web page performance. Our end goal is to identify critical bottlenecks and extraneous components in the page load process with respect to the hardware. Our work is especially focussed on addressing the problems in lower-end mobile devices that are popular in developing regions.

REFERENCES

[1] Werner Almesberger. 1998. *Linux traffic control.* Technical Report.
[2] http://hwstats.unity3d.com/mobile/. 2017. (2017).
[3] https://developer.chrome.com/devtools. 2017. (2017).
[4] Javad Nejati and Aruna Balasubramanian. 2016. An in-depth study of mobile browser performance. In *Proc. WWW 2016*. 1305–1315.
[5] Qualcomm Development Network. 2017. developer.qualcomm.com/software/snapdragon-profiler. (2017).
[6] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. 2013. Demystifying Page Load Performance with WProf.. In *NSDI*. 473–485.
[7] Yuhao Zhu and Vijay Janapa Reddi. 2017. Optimizing General-Purpose CPUs for Energy-Efficient Mobile Web Computing. *ACM Transactions on Computer Systems (TOCS)* 35, 1 (2017), 1.