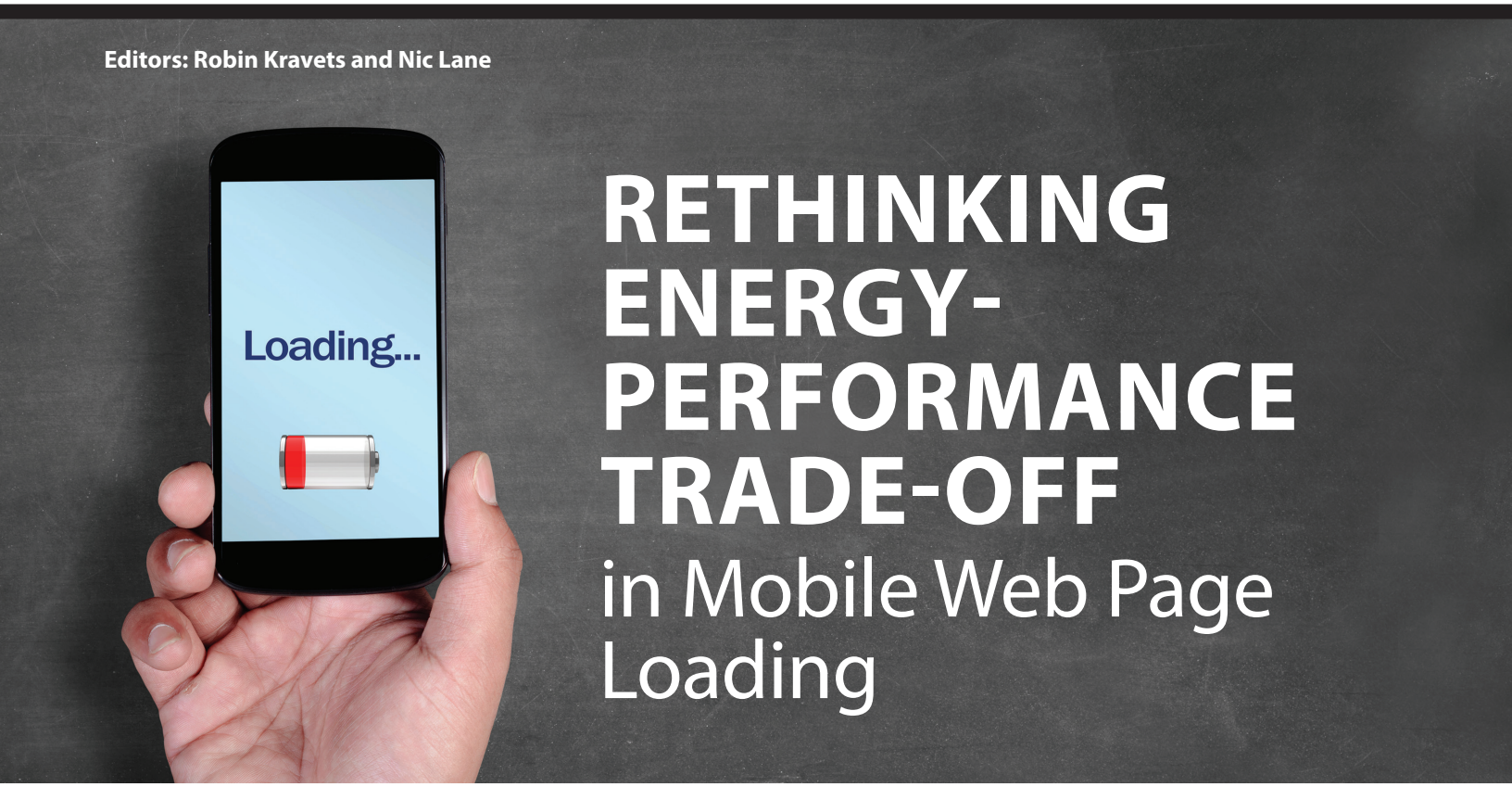
Duc Hoang Bui *KAIST, Daejeon, South Korea* **Yunxin Liu** *Microsoft Research, Beijing, China*
Hyosu Kim and Insik Shin *KAIST, Daejeon, South Korea* **Feng Zhao** *Microsoft Research, Beijing, China*

Editors: Robin Kravets and Nic Lane



RETHINKING ENERGY-PERFORMANCE TRADE-OFF in Mobile Web Page Loading

Excerpted from "Rethinking Energy-Performance Trade-Off in Mobile Web Page Loading," from *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking* with permission. http://dl.acm.org/citation.cfm?id=2790103 © ACM 2015

Web browsers are one of the core applications on smartphones and other mobile devices, such as tablets. However, web browsing, particularly web page loading, is of high energy consumption as mobile browsers are largely optimized for performance and thus impose a significant burden on power-hungry mobile devices. With the advent of modern web capabilities, websites even become more complex and energy demanding. In the meantime, slow progress in battery technology constrains battery budget for mobile devices. As users are more aware of energy consumption of apps, it is desirable to improve the energy efficiency of web browsing, particularly web page loading.

Although minimizing energy is necessary, it is essential to reduce energy without degrading user experience. An important reason is that slow loading time can have a big cost for businesses. For example, every 100ms delay costs 1% of sales for Amazon [1] or 1-second delay in Bing search engine results in a 2.8% drop in revenue per user [2]. "As users migrate to mobile, page load time is perhaps the most important metric we have," said Howard Mittman, VP and publisher of *GQ* magazine [3].

We seek to reduce the energy consumption of web page loading on smartphones without compromising user experience[1] [4]. In particular, we aim to not increase page load time. To achieve this goal, we study browser internals and system behaviors to understand how the energy is spent in loading web pages, and to identify opportunities to improve the energy efficiency. The main architecture of Chrome web browser is illustrated in Figure 1.

Although many browser manufacturers have made an effort to improve energy efficiency for mobile devices [5], our findings indicate that the current mobile browsers are not yet fully energy optimized for web page loading. First, the web resource processing is aggressively conducted regardless of contents and network conditions at the risk of energy inefficiency. Second, the content painting rate is unnecessarily high, consuming a lot of energy without bringing user-

[1] Video demos and source code of this paper are available at cps.kaist.ac.kr/eBrowser

Photo, istockphoto.com

perceivable benefits. Finally, the power-saving capability of modern CPUs with the emerging ARM big.LITTLE architecture [6] is underutilized. Fundamentally, the web page loading is overly optimized for performance but not for energy cost.

We argue that the energy-performance trade-off must be reconsidered for web page loading on mobile devices. Based on our findings, we formulate new design principles for energy-efficient web page loading. Based on these principles, we develop three new techniques, each one addressing one of the above energy-inefficiency issues. First, we propose to use the *network-aware resource processing* (NRP) technique to effectively trade performance for energy reduction by adapting to changing network conditions. We use adaptive resource buffering to control the speed of web resource processing dynamically with regard to the speed of resource download, in order to become energy efficient without increasing page load time. Second, we propose the *adaptive content painting* (ACP) technique to avoid unnecessary content paints to reduce the energy overhead. We study the trade-offs between energy saving and the increase of page load time to ensure the user experience is not compromised.

Finally, to better leverage the big.LITTLE architecture, we propose the *application-assisted scheduling* (AAS) technique to leverage internal knowledge of the browser to make better scheduling decisions. Specifically, we employ adaptive thread scheduling based on Quality-of-Service (QoS) feedback in a way that the browser keeps threads running on *little* cores to save energy, as long as their QoS requirements are being satisfied.

Energy-efficient mobile web browsing has been explored through various approaches. Our work leverages browser internals (e.g., process/thread structure and resource fetching/processing pipelines) for energy-efficient browsing, while others focus on the characteristics of web pages (e.g., primitives [7], colors [8] and network accesses [9]). Thereby, we believe our work can be integrated with others to improve energy efficiency further.

ENERGY-EFFICIENT PAGE LOADING

Network-Aware Web Resource Processing: Although immediately processing data received from the server is natural and minimizes page load time, it is not energy efficient due to the accumulative overhead of the processing of small data chunks.

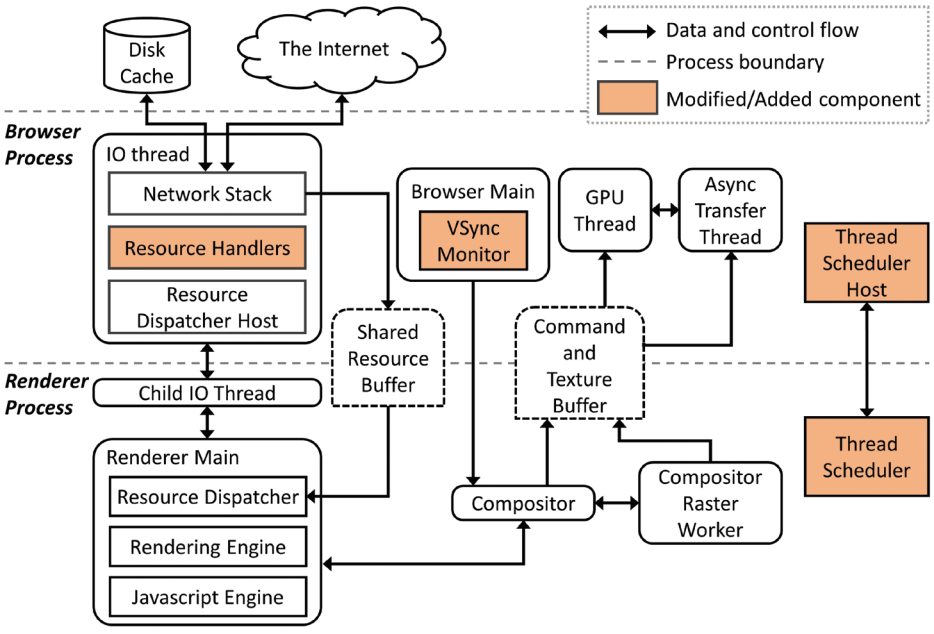
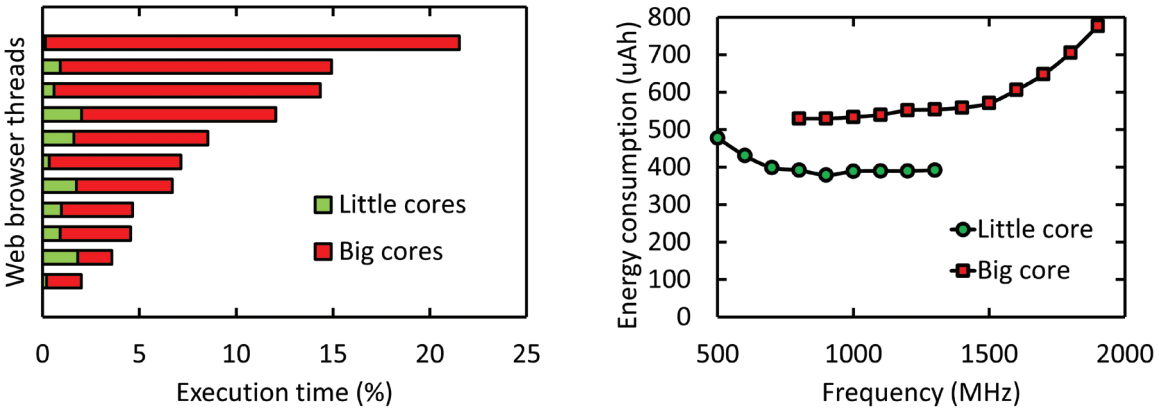


FIGURE 1. Architecture of Chrome web browser

Each time the *read* system call returns a data buffer, even if it is small, the Renderer process has to go through the whole web page rendering pipeline. In particular, for image data, many graphic activities are involved in the Compositor, Raster Worker and Async Transfer threads. As a result, the accumulated overheads are high and thus waste much energy, more than 10% of the total system energy cost [4].

The choice of how often to conduct web resource processing influences performance and energy efficiency. Frequent resource processing may come with energy inefficiency, since it can yield marginal progress while consuming a significant amount of energy. Therefore, we propose to design the web resource processing adaptive to the speed of downloading web resources, which we refer to as *Network-aware Resource Processing* (NRP). A basic underlying principle is that the faster downloading, the larger batch size for better energy-efficiency should be. That is, we have a smaller batch size for slower downloading so as not to introduce excessive delays in web resource processing. We choose the network throughput (rate of receiving data) as an indirect but low-overhead indicator of the changes of the displayed content instead of doing accurate-yet-heavy analyses on user-perceived content changes on the screen for web resources received. Furthermore, to reduce the impact of NRP on page load time, we do not apply the buffering to critical resources that may delay the downloading of other resources.

Adaptive Content Painting: Content painting (i.e., web page rendering and screen updating) comes with a trade-off between the user experience and energy overheads. Although a high frame rate can provide very smooth user experience, it can incur a high overhead to GPU and CPU to render web pages and, when combined with high resolution, consume a lot of energy [10]. Content paints are not equal but may introduce different degrees of change on the screen. As the majority of content paints generate a zero or very small visible screen change, multiple content paints can be aggregated together to save energy without compromising user experience. For example, when Chrome loads the top 10 websites in the United



(a) Execution time of threads in Chrome browser

(b) Energy consumption of cores on Samsung Galaxy 55 G900H

FIGURE 2. Energy efficiency and utilization on big.LITTLE architecture

States, we found that 56% of the paints do not generate any visible screen change, and 62% of the paints generate a visible screen change of less than 5%.

Since the design of high-rate content painting is overly optimized for performance, not energy efficiency, we design the content painting to be adaptive to the visible changes on the screen, which we refer to as *Adaptive Content Painting* (ACP). We introduce a new parameter, called *paint_rate*, that limits the rate of content painting and dynamically adapts to the content changing speed so that we can aggregate content painting to save energy while preserving user experience. The *paint_rate* parameter increases when the content changes fast, and vice versa, within a predetermined range. Ideally, ACP technique should quantify the visual changes between painted frames to adapt with the degree of changes of content. However, doing so requires comparing consecutive frames pixel by pixel, which imposes heavy computation and thus a high energy cost. Therefore, we use a lightweight approach of linearly increasing and fast decreasing the *paint_rate* parameter, without any extra computation cost [4].

Application-Assisted Scheduling: Although big.LITTLE architecture is designed to save energy, we find that the energy-saving potential of big.LITTLE is not fully exploited in the case of Chrome. For example, as shown in Figure 2(a), when loading *instagram.com*, 89% of the total execution time is on the big cores. Specifically, we find that the little cores are underutilized. The reason of low little-core utilization is that the OS schedules threads based on a load-driven approach that favors performance over energy saving. On symmetric multi-core architectures, this approach is also energy efficient because if a thread can finish its task faster, the CPU can go to sleep sooner [11]. However, on heterogeneous multi-core architectures like big.LITTLE, finishing a thread sooner may not reduce the energy cost. As shown in Figure 2(b), a little core has a lower energy per instruction cost than a big core. Thus, a thread should run on a little core as long as it can tolerate the resulting delay.

To better utilize big.LITTLE architecture to save energy, we propose to leverage internal knowledge of browsers for energy-efficient scheduling. We allow browsers to decide whether a thread should run on a big or little core. Browsers know much more information about their threads than the OS scheduler, e.g., what type of tasks the threads do, how important the threads are, how long finishing time a thread can tolerate, the semantics of the threads, and the relationship among them. Therefore, browsers may make better decisions on assigning threads to big or little cores. We design the AAS technique adaptive to QoS, as QoS is critical to user experience. The QoS is estimated by the frame rate of the browser. Thereby, the thread scheduling decisions are determined based on fine-grained QoS instead of CPU load. We call this kind of technique *Application-Assisted Scheduling* (AAS).

IMPLEMENTATION

We have implemented the three techniques on Android by modifying Chromium (the open source project of Chrome browser) version 38, which was the latest stable version when we conducted the work of this paper. In total, we add about 1,200 lines of code into various modules of the browser as highlighted on Figure 1. No rooting or modification OS kernel and websites are needed.

EVALUATION

We present here the results for Samsung Galaxy S5 G900H model that uses Samsung Exynos 5422 SoC with 4 Cortex-A15 big cores and 4 Cortex-A7 little cores. The smartphone runs Android 4.4.2 on Linux kernel 3.10. We use a Monsoon power monitor to measure the total system energy consumption of the smartphones. We use the top 100 websites in the United States according to Alexa in May 2014. A user study and results on other configurations with different metrics can be found in our paper [4].

Total energy saving: Figure 3(a) shows how much total energy saving can be achieved by all the three techniques together. The average total energy saving is 24.4%, ranging from 0.02% to 66.5%. 62 websites

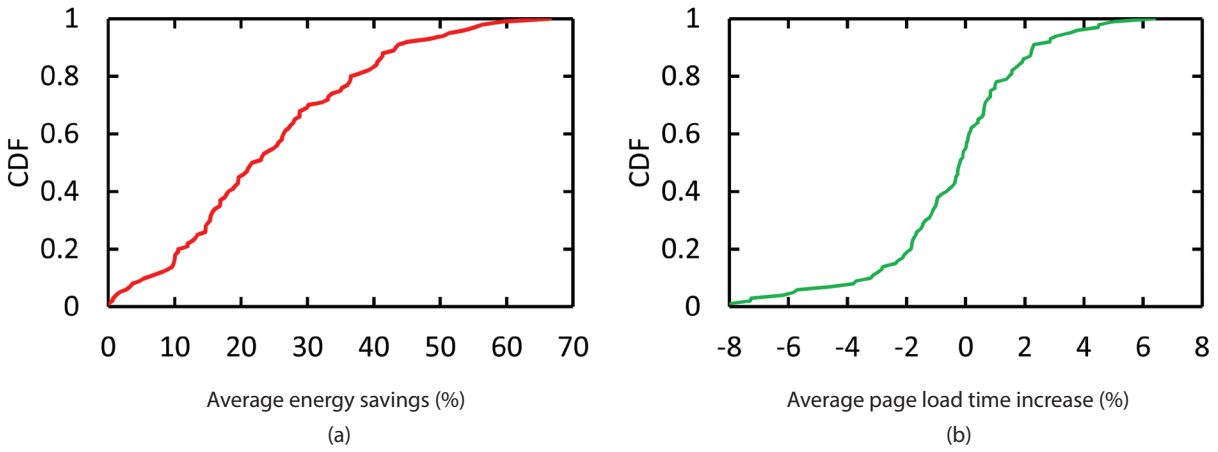


FIGURE 3. Total energy saving and PLT increase.

have an energy saving of 15%-45%, and seven websites have an energy saving of more than 50%. These results demonstrate that our techniques are able to significantly reduce the energy cost of web page loading. The most energy reduced (66.5%) website is *infusionsoft.com* that has a heavy slide show of high resolution images with a fading transition effect. The least energy reduced websites are the ones with simple text content, such as *usps.com*. The most effective technique is AAS, reducing the average energy consumption by 19.5%.

Total page load time increase: Our techniques impose minimal extra page load time (PLT) as shown on Figure 3(b). The average PLT is even decreased by 0.38% (or decrease 29 ms in terms of the absolute PLT), ranging from -8.11% to 6.38%. This indicates that our techniques can reduce unnecessary workload, allowing web pages to load faster. In fact, 55 websites have a decreased PLT and 93 websites have PLT increases less than 3%. In terms of the absolute PLT, 94 websites have PLT increases less than 0.2 seconds.

CONCLUSION

This paper presented three effective techniques to optimize the energy consumption of web page loading on smartphones. Two of the techniques, network-aware resource processing and adaptive content painting, are designed to address energy-inefficiency issues of the current mobile web browsers in its content processing and graphic processing pipelines. The third one, application-assisted scheduling, is designed to balance the trade-off between the energy saving and the QoS on big.LITTLE platforms. We have implemented the proposed techniques on Chromium and Firefox, and conducted comprehensive evaluations using real-world websites and latest-generation smartphones. Experimental results and user study show that the techniques are able to significantly reduce the energy cost of web page loading and introduce hardly perceivable page load time increase. ■

Duc Hoang Bui is a PhD student in School of Computing at KAIST, Korea, where he received an M.S. in computer science. He was an intern at Samsung Electronics and Microsoft Research Asia. Now his research focuses on improving the performance and energy efficiency of web browsers on mobile systems.

Dr. Yunxin Liu is a lead researcher at System Group, Microsoft Research Asia. He received his Ph.D. degree in Computer Science from Shanghai Jiao Tong University. His research interests are mobile systems and networking, focusing on power management, security and privacy, and human sensing.

Hyosu Kim is a Ph.D. candidate in School of Computing at KAIST, South Korea. He received a B.S. degree from Sungkyunkwan University, Korea, and a M.S. degree from KAIST, Korea in 2012. His research interests are in the areas of mobile computing and sound engineering.

Insik Shin is an associate professor in the School of Computing Science at KAIST. He obtained a Ph.D. from University of Pennsylvania. His research interests lie in mobile computing, real-time embedded systems, and cyber-physical systems. He received Best Paper Awards from various conferences, including RTSS and RTAS.

Dr. Feng Zhao is the chief technology officer and vice-president for Advanced R&D and Smart Home business at Haier. He received his Ph.D. degree in Electrical Engineering and Computer Science from MIT. He is an IEEE fellow and his research has focused on networked embedded systems.

REFERENCES

[1] G. Linden, "Make Data Useful," 2006.
[2] E. Schurman and J. Brutlag, "The User and Business Impact of Server Delays, Additional Bytes, and HTTP Chunking in Web Search," in *O'Reilly Velocity Web Performance and Operations Conference*, 2009.
[3] T. Everts, Time Is Money: The Business Value of Web Performance, O'Reilly, 2016.
[4] D. H. Bui, Y. Liu, H. Kim, I. Shin and F. Zhao, "Rethinking Energy-Performance Trade-Off in Mobile Web Page Loading," in *ACM MobiCom*, 2015.
[5] B. Heenan, "Building a more power efficient browser," *Microsoft Edge Dev Blog*, 2016.
[6] "big.LITTLE Technology," ARM Ltd., 2016. [Online]. Available: http://www.thinkbiglittle.com/.
[7] N. Thiagarajan, G. Aggarwal and A. Nicoara, "Who Killed My Battery: Analyzing Mobile Browser Energy Consumption," in *WWW*, 2012.
[8] M. Dong and L. Zhong, "Chameleon: A Color-Adaptive Web Browser for Mobile OLED Displays," in *ACM MobiSys*, 2011.
[9] B. Zhao, W. Hu, Q. Zheng and G. Cao, "Energy-Aware Web Browsing on Smartphones," *IEEE Trans. on Parallel and Distributed Systems*, 2015.
[10] K. W. Nixon, X. Chen, H. Zhou, Y. Liu and C. Yiran, "Mobile GPU Power Consumption Reduction via Dynamic Resolution and Frame Rate Scaling," in *USENIX HotPower*, 2014.
[11] A. Carroll and G. Heiser, "Mobile Multicores: Use Them or Waste Them," in *USENIX HotPower*, 2013.