Exploiting Webpage Characteristics for Energy-Efficient Mobile Web Browsing

Yuhao Zhu, Aditya Srikanth, Jingwen Leng, Vijay Janapa Reddi *Member, IEEE*
Department of Electrical and Computer Engineering
The University of Texas at Austin
{yzhu, aditya.srik, jingwen}@utexas.edu, vj@ece.utexas.edu

***Abstract*—Web browsing on mobile devices is undoubtedly the future. However, with the increasing complexity of webpages, the mobile device's computation capability and energy consumption become major pitfalls for a satisfactory user experience. In this paper, we propose a mechanism to effectively leverage processor frequency scaling in order to balance the performance and energy consumption of mobile web browsing. This mechanism explores the performance and energy tradeoff in webpage loading, and schedules webpage loading according to the webpages' characteristics, using the different frequencies. The proposed solution achieves 20.3% energy saving compared to the performance mode, and improves webpage loading performance by 37.1% compared to the battery saving mode.**

***Index Terms*—Energy, EDP, Cutoff, Performance, Webpages**

I. Introduction

Web experience is becoming more interactive, collaborative, and personalized in today's world. Two significant trends are becoming clear. First, web browsing on mobile devices will be dominant in the near future. International Data Corporation (IDC) forecasts that by 2015, there will be more mobile devices than humans on the planet, and web surfing with mobile devices will surpass that on desktop environments [6]. Thus, it is paramount that web browsing is well supported on mobile platforms, regardless of whether the device is a smartphone, tablet, or laptop computer.

Second, webpages have become more computationally intensive and energy consuming over the years. For the 50 hottest websites listed on `www.alexa.com`, we measured the network transmission time and page-content processing time over the past 11 years on the Pandaboard ES (based on a dual-core ARM Cortex-A9 processor connected to the 100 Mb/s Ethernet). Randomly picking one image archived by Internet Archive [4] from each year, we found that `www.cnn.com` captures the most representative trend among all these hot websites. Although there is about 5% measured overlapping time between network and computation, the overall gap between network and computation time has been significantly and consistently increasing over the years, as shown in Fig. 1. The fitted line represents this consistent trend, which quantifies a tenfold relative increase in webpage computation intensity from the perspective of compute versus network. We observe a similar trend among other popular websites, and as such believe that the browser workload is steadily becoming compute bound and will therefore have more impact on a device's energy consumption. This trend is likely to hold true in the future with the adoption of new web standards and technologies such as HTML5 and WebGL, which require more demanding computational processing capabilities. This trend will become especially more concerning in the mobile era where energy efficiency is a first-class citizen.

Manuscript submitted: 27-Jun-2012. Manuscript accepted: 24-Aug-2012. Final manuscript received: 31-Aug-2012.

There are a few proposals such as [10], [7] that aim to improve web browser performance. Most are focused on parallelizing browser-specific tasks such as parsing, CSS selection, etc. Although these parallelized algorithms can achieve speedups ranging from 4x to 80x for various browsing tasks [10], they do not typically scale well [10], making the parallelization methodology less favorable.

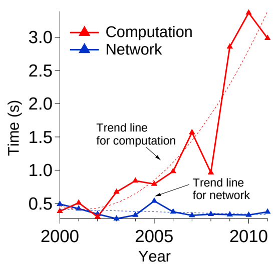


Fig. 1: Webpage computation time has increased significantly.

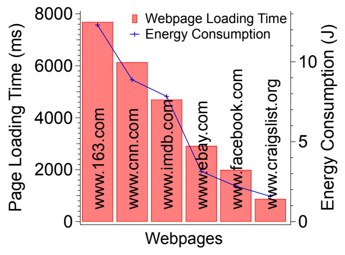


Fig. 2: Webpages vary in performance and energy needs.

On the other hand, heterogeneous systems with different core configurations each with different P-states provide a unique opportunity that enables both high performance and energy efficiency. We believe that fully harnessing such systems entails detailed characterization and quantification of significantly varying performance and energy consumption across different webpage loadings. Fig. 2 confirms the presence of such different loading behaviors "in the wild" across five randomly chosen websites. To enable research in that direction, we take a first step and mainly explore dynamic voltage and frequency scaling (DVFS) available in most systems. We make the following contributions in this paper toward our objective:

- We *characterize the webpage-level differences* of web browser page loading, which leads to novel insights on how webpage differences can be applied to achieve energy-efficient mobile web browsing.
- On the basis of the webpage characterization analytics, we derive regression models for *predicting the performance and energy consumption of webpage loading*.
- Using the prediction model, combined with *intelligent scheduling of webpages using different frequencies in the processor*, we achieve 20.3% energy improvement and 37.1% performance improvement compared to the native performance and battery saving mode, respectively.

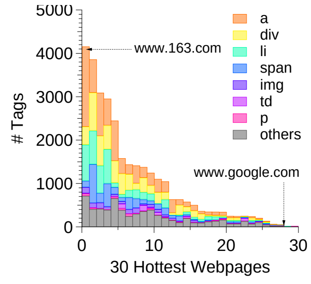
II. Webpage-Level Characterization

The prerequisite to addressing the performance and energy issues for computationally intensive webpages is the thorough understanding of web-browsing workloads. Previous works [8], [9] treat the web browser as the target application and perform system- and microarchitecture-level characterizations/optimizations while treating webpages as input sets to the target application. However, webpages contain richer information than input sets in the traditional sense. Webpages bear full program semantics, put inherent constraints in the processing order, and may even modify themselves (JavaScript). Motivated by such information, we profiled the hottest 30 webpages listed in `www.alexa.com`, analyzed webpages as full-duty workloads, and identified differences across webpages to explore energy-efficiency opportunities. The way webpages are developed permits analysis along two dimensions: *static*, which includes HTML and CSS, and *dynamic* (i.e., JavaScript).

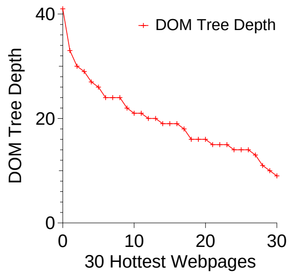
A. Hypertext Markup Language (HTML)

HTML uses *tags*, each possibly associated with zero or more *attributes*, combined with plain text to describe the webpage's overall structure. Along with parsing the HTML file, the rendering engine also constructs the *DOM* (Document Object Model) *tree* structure dynamically with each node in the tree corresponding to an HTML tag. The complex and irregular tree-related computations indicate that the difference in tags, attributes, and DOM tree structures can lead to different processing overhead and thus different energy implications.

We performed a cumulative distribution analysis of the HTML tag usage. About 10% of the possible tags make up 90% of tag usage, mainly because web developers tend to use a few common tags. In order to know what these hot tags are, we performed a tag distribution profiling, shown in Fig. 3a sorted by the total number of tags. We identify a few tags that are hot across all the webpages and lump the rest together in the plot. As the figure shows, the number of tags varies significantly across different webpages. For instance, `www.baidu.com`, which is the 5^{th} hottest website, only has 70 tags while `www.163.com`, which ranks 28^{th}, has 4,135 tags. It is interesting to note that hot tags are not always hot in all webpages; `www.baidu.com` does not have the <td> tag. These indicate strong "interpage" differences.



(a) HTML tag distributions.



(b) DOM tree depths.

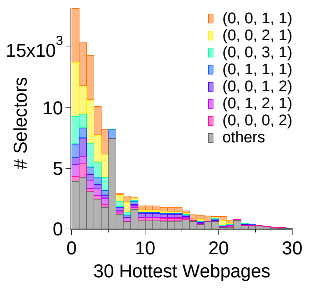
Fig. 3: HTML analysis indicates that tags and the complexity of the DOM tree are significantly different across webpages.

We performed similar profiling on attributes and observed similar interpage differences as in tags. For instance, we observed that the number of attributes in the hottest 30 webpages can vary from 9 to 8,351, with a relative standard deviation of 129.83%. Such variances can impact performance and energy usage significantly when the rendering engine generates the webpage's visual properties.

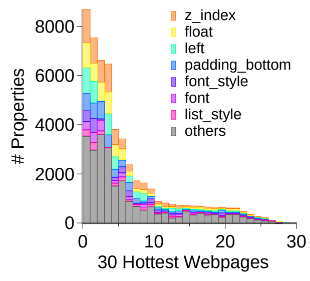
We also capture the DOM tree structure's complexity according to the tree depth. Apart from HTML processing, even CSS processing relies heavily on the DOM tree structure to perform tasks such as selector matching, which was found to be one of the most computation-intensive parts in web browsers. Fig. 3b sorts the hottest 30 webpages by the tree depth. As shown, DOM tree depth is significantly different across webpages, with a relative standard deviation of 33%, indicating strong interpage differences. Such differences lead to different DOM-tree-related processing overheads that affect performance and energy consumption of webpage loading.

B. Cascading Style Sheets (CSS)

CSS dictates how HTML tags are displayed using a set of *rules*, each with a *selector* and a set of visual *properties*. Referring to the DOM tree structure, the rendering engine applies CSS rules to each HTML element and gives exact coordinates in order to lay out the webpage. These CSS-related processing tasks such as selector matching, style computation, etc. are known for their computational intensity [10].



(a) CSS selectors.



(b) CSS properties.

Fig. 4: CSS analysis indicates that both selector and property values are used very differently across several webpages, which leads to different computation and energy requirements.

In order to capture the complexity of processing CSS rules, we choose to use *specificity* [1] as the metric. Specificity is a 4-tuple, with each element describing one pattern of CSS selectors. We observed a similar selector specificities distribution in external CSS selectors (shown in Fig. 4a) as HTML tags. In particular, although there are a few hot selector specificities across all webpages, several webpages still rely heavily on their own selector patterns. This interpage difference affects how selectors are matched to HTML tags, leading to different performance and energy requirements. Furthermore, we profiled CSS property usage across different webpages. Similar to HTML tags, in Fig. 4b we observe hot properties and significantly varied usage of these properties across webpages. Such a difference in CSS properties translates to different performance and energy behaviors during the style computation phase, where the rendering engine uses CSS properties to resolve the visual and layout information for the entire webpage.

C. JavaScript

HTML and CSS define the static behavior of webpage loading. However, webpages may embed JavaScript (JS) programs that control a webpage's dynamic behavior after the initial webpage load. In general, a JS program is a collection of event handlers. They are triggered by webpage internal state transitioning or user actions such as mouse click, and most often happen after the webpage's basic content is rendered. These event handlers change webpages dynamically through *DOM methods*, and retrigger operations such as DOM tree construction, layout, etc. that are originally performed during the initial load.

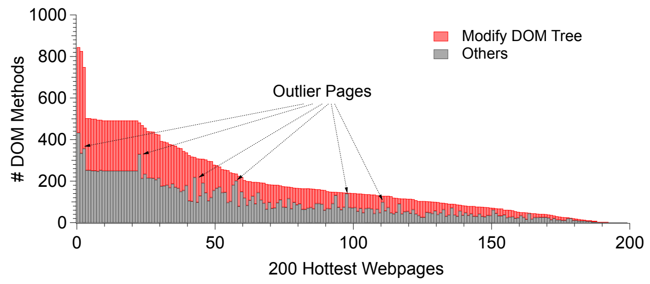


Fig. 5: Distribution of JS methods implies different JS-related computation complexities across different webpages.

We crawled the hottest 200 webpages listed on `alexa.com` and counted the number of DOM methods across the webpages. Fig. 5 shows this statistic sorted by the total number of DOM methods. We classify DOM methods as methods that change the DOM tree topology, and methods that do not modify the DOM tree structure. The latter is shown as Others in the figure. Overall, most DOM methods used in real webpages modify the DOM tree, therefore retriggering basic rendering operations that could lead to intensive computations when complex DOM tree structures are involved. In addition, interpage variance is also observed. Although most webpages contain DOM-tree-related operations, there are a few exceptions (such as those pointed out in Fig. 5) where most dynamic operations trigged by JavaScript events do not touch the DOM tree structure, indicating a relatively lower performance and energy overhead.

III. Webpage Performance and Energy Modeling

Mobile systems are more constrained by battery life than power. Therefore, we choose energy, in addition to performance, to evaluate browser-like workloads. We relate performance and energy usage of webpage loading to the webpage-level characterization performed in the previous section, and quantify that by applying regression modeling on webpage-level characterization results to derive a statistical inference model.

A. Model Derivation

We use 200 webpages to obtain the training set. Each measurement in the training set maps a set of meta-information to observed per-page loading time and energy consumption. This data is then used to train linear regression models that fit and predict the performance and energy consumption.

Table I identifies the model features and groups them as HTML and CSS. Using domain-specific knowledge, we prune features that rarely appear and that have negligible performance impact according to microbenchmarking. We also find that features describing the same webpage characteristics tend to have strong correlation with each other. We then cluster features by calculating the correlation matrix, and pick only one feature from each cluster.

TABLE I: Model Features

Groups	Model Features
HTML	Number of each tag Number of each attribute Number of DOM tree nodes DOM tree depth
CSS	Number of rules Average selector specificity Number of each property

We use R and its packages to perform linear regression with regularization. We also perform logarithmic transformation on features and responses (loading time and energy) before doing a fit to reduce the value magnitude and improve linearity.

B. Model Evaluation

For validating the model, we performed experiments on the Pandaboard ES (rev. B1) running a preinstalled OMAP4 image of Ubuntu 12.04 release with Linux kernel version 3.2.14. The Pandaboard is equipped with a dual-core Cortex-A9 OMAP4460 and ran at 1.2 GHz in this experiment. We instrumented Mozilla Firefox 12.0 source code for loading time and energy measurements, such that only "page loading" is instrumented without considering Firefox bootstrapping and shutdown. In order to focus on studying the webpage itself, we also always disable browser cache and isolate the effects of (partially) caching webpage resources in both browser and hardware. In addition, motivated by Fig. 1, we focus on computation and isolate network and disk overhead by downloading the webpages and mapping them into memory.

To measure the webpage loading time, we instrumented Firefox and inserted signatures at the start and end of loading a webpage and recorded their respective timestamps. For energy measurements, we built a custom power sensing circuitry, using a sense resistor located between the off-chip voltage regulator module (VRM) and V_{DD} to the chip [5]. Using National Instruments' data acquisition unit 6133 we gathered power measurements at a rate of 100,000 samples per second.

In addition to the 200 webpages used to train the model, we obtained 800 additional webpages listed on `www.alexa.com` for evaluating the regression models. Overall, the performance and energy models have a mean error rate of 5.5% and 4.6%, respectively. Cumulative distribution analysis of the error rates shows that the performance and energy model can predict 87% and 89% of webpages within 10% error, respectively.

IV. Webpage Loading: Energy-Delay Tradeoff

Performance and energy-prediction models derived in the previous section provide a unique opportunity to achieve high-performance and energy-efficient mobile web browsing.

Asymmetric systems such as NVIDIA's Tegra 3 [2] incorporate both high-frequency, energy-consuming and low-frequency, energy-conserving cores, and thus provide a large optimization space for exploring the tradeoff between performance and energy consumption. We believe that applying our inference models for webpage scheduling helps us efficiently utilize heterogeneous resources. As a first step, in this section we use the DVFS knob available on Pandaboard to mimic the scheduling on such heterogeneous systems. We focus only on the static loading of webpages and leave the dynamic behavior caused by JS activity as opportunity for future work.

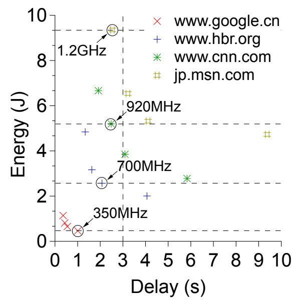
Mobile systems must make tradeoffs between energy (E) and delay (D) while accounting for the webpage-loading cutoff latency. We identify two typical operating modes in mobile systems that make E-D tradeoff differently. In particular, the battery-saving mode aims at minimizing energy consumption and uses the lowest frequency of the processor. The performance mode intends to deliver webpages as fast as possible, and therefore uses the highest frequency of the processor.

In order to understand the E-D tradeoff, we measured the loading time and energy consumption of the hottest 1,000 webpages listed by `www.alexa.com` on the Pandaboard. The Cortex-A9 processors on the Pandaboard support four frequency levels: 350 MHz, 700 MHz, 920 MHz, and 1.2 GHz. We swept all frequencies and chose 3s as the cutoff latency for webpage loading according to the well-known "three-second rule" [3]. Fig. 6a shows four representative webpages having different optimal frequencies that meet the cutoff latency while achieving the minimal energy consumption.

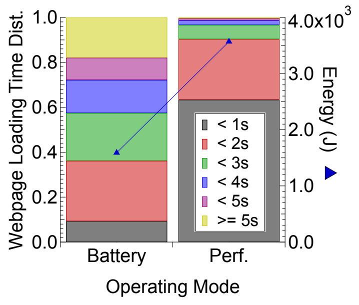
A fixed frequency is not necessarily the best choice for all the webpages. As Fig. 6a shows, simple websitesig such as `www.google.cn` and `www.hbr.org` can be loaded using 350 MHz and 700 MHz to meet the cutoff latency, whereas more complex websites such as `www.cnn.com` and `jp.msn.com` require higher computation capabilities from higher frequencies. In general, lower frequencies result in lower processor energy consumption, and thus suggest the energy saving opportunity at load time using lower frequencies without violating the cutoff latency.

Fig. 6b shows the distribution of webpage loading time under each mode. Each region represents the portion of webpages that meet a particular cutoff latency. We also show the total energy consumption of 1,000 webpages under each mode on the right y-axis. Assuming a 3s cutoff latency, the battery mode consumes 1.6 kJ of energy, 55.5% less than the performance mode; however, it violates the cutoff latency for 42.6% of webpages, more than the performance mode which only violates 3.3% of webpages. This extreme imbalance requires a mechanism that balances E and D more effectively.

The performance and energy models we derived in the previous section, however, can accurately predict the optimal frequency that meets the cutoff latency while minimizing energy. After the webpage is parsed, which according to our task profiling is only 1% of the entire time, our models can quickly predict the loading time and energy consumption and then decide the optimal frequency. We exert our scheduling technique; it achieves 20.3% energy savings, or 0.73 J saving per webpage, as compared to the performance mode, while



(a) Optimal frequency to meet cut-off while minimizing energy.



(b) Distribution of webpages loading time under two operating modes.

Fig. 6: A fixed frequency does not work best for all webpages.

only violating 2.2% more webpages. Meanwhile, it reduces the number of violated webpages by 37.1% compared to the battery mode, with 78.8% more energy consumption.

We also compared our scheduling mechanism with a "static balanced mode" that always loads webpages using 700 MHz. Under the 3s cutoff latency, we reduced 5% of cutoff violations by consuming 1% less energy. As we tightened the cutoff latency to 2s, our scheduling mechanism reduced violations by 16% compared to the static mode with only 6% more energy consumption.

V. Conclusion

We proposed an innovative scheme to achieve energy-efficient mobile web browsing. Leveraging the insight that different webpages have varying performance and energy consumption, we derived statistical inference models to accurately predict the loading time and energy consumption for each webpage. Building on such models, we explored the webpage loading cutoff latency to balance energy consumption and user browsing experience by scheduling webpage loading using processor DVFS. Measured hardware results show that our technique reduces energy consumption by 20.3% compared to the performance mode, and it improves webpage loading performance by 37.1% as compared to the battery mode.

References

[1] *CSS selector specificity*. [Online]. Available: http://www.w3.org/TR/CSS21/cascade.html#specificity

[2] *NVIDIA Tegra 3*. [Online]. Available: http://www.nvidia.com/object/tegra-3-processor.html

[3] *The Three Second Rule*, Std. [Online]. Available: http://rd2inc.com/blog/2012/04/the-three-second-rule/

[4] *Wayback Machine Internet Archive*. [Online]. Available: http://archive.org/web/web.php

[5] *Pandaboard ES Rev B1 Schematic*, Oct, 2011. [Online]. Available: http://pandaboard.org/sites/default/files/board_reference/pandaboard-es-b/panda-es-b-schematic.pdf

[6] *More Mobile Internet Users Than Wireline Users in the U.S. by 2015*, Std., Sep, 2011. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS23028711

[7] C. Badea, M. R. Haghighat, A. Nicolau, and A. V. Veidenbaum, "Towards parallelizing the layout engine of firefox," in *Proc. of the USENIX conference on Hot topics in parallelism*, 2010.

[8] V. Bhatt, N. Goulding-Hotta, Q. Zheng, J. Sampson, S. Swanson, and M. B. Taylor, "Sichrome: Mobile web browsing in hardware to save energy," in *Dark Silicon Workshop, ISCA*, 2012.

[9] A. Gutierrez, R. Dreslinski, A. Saidi, C. Emmons, N. Paver, T. Wenisch, and T. Mudge, "Full-system analysis and characterization of interactive smartphone applications," in *Proc. of IISWC*, 2011.

[10] L. A. Meyerovich and R. Bodik, "Fast and parallel webpage layout," in *Proc. of WWW*, 2010.