View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Lancaster E-Prints

Adaptive Web Browsing on Mobile Heterogeneous Multi-cores

Jie Ren†, Xiaoming Wang†, Jianbin Fang‡, Yansong Feng§, Zheng Wang*
† Shaanxi Normal University, China, ‡National University of Defense Technology, China, § Peking University, China
*Lancaster University, United Kingdom

***Abstract*—Web browsing is an important application domain, but it imposes a significant power burden on mobile devices. While heterogeneous multi-cores offer the potential for energy-efficient computing, existing web browsers fail to exploit the hardware to optimize mobile web browsing. Our work aims to offer a better way to optimize web browsing on heterogeneous mobile devices. We achieve this by developing a machine learning based approach to predict the optimal processor setting for rendering the web content. The prediction is based on the web content, the network status and the optimization goal. We evaluate our approach by applying it to the Chromium browser and testing it on a representative big.LITTLE mobile platform. We apply our approach to the top 1,000 hottest websites across seven typical networking environments. Our approach achieves over 80% of the performance delivered by a perfect predictor. Our approach achieves over 30%, 50%, and 60% improvement respectively for load time, energy consumption and the energy delay product when compared to two state-of-the-art approaches.**

I. Introduction

Web is a major information portal on mobile devices [1]. Yet, web browsing is poorly optimized and continues to consume a significant portion of battery power on mobile devices [2]. Heterogeneous multi-cores offer a new way for energy-efficient mobile computing. However, current mobile web browsers rely on the operating system (OS) to exploit the heterogeneous cores. Since the OS has little knowledge of the web workload and the network conditions, its decision is often sub-optimal. This leads to poor energy efficiency [3], draining the battery faster than necessary and irritating mobile users.

Our work aims to provide a better way for optimizing mobile web browsing. Rather than letting the OS make all the scheduling decisions by passively observing the system's load, we want the browser to decide which heterogeneous core and the optimal clock frequencies to run the rendering engine. We believe such a decision must consider the web content, the optimization goal, and how the network affects the rendering process. We achieve our goal by employing machine learning techniques to automatically build predictors based on empirical observations gathered from a set of training examples. The trained models are then used at runtime to predict the optimal processor configuration for any *unseen* webpage.

We present a machine-learning-based web rendering scheduler that can leverage knowledge of the network and webpages. We compare our approach against two state-of-the-art approaches [4], [5] on a representative big.LITTLE mobile platform. Experimental results show that our approach outperforms the state-of-the arts by delivering over 1.3x improvement across evaluation metrics.

II. Motivation

Consider a scenario for browsing four webpages, starting from the home page of `news.bbc.co.uk` on an Odriod Xu3 big.LITTLE mobile platform with a Cortex-A15 (big) and a Cortex-A7 (little) processors (see also Section IV-A).

Networking environments. We consider three typical networking environments: Regular 3G, Regular 4G and WiFi (Section III-B1). To ensure reproducibility, web requests and responses are deterministically replayed. The web server simulates the download speed and latency of a given network setting.

Scheduling policies. We schedule the Chromium rendering engine to run on either the big or the little core under different clock frequencies to find the best processor configuration per test case. We refer this best-found configuration as the `oracle` performance. We use the `interactive` CPU frequency governor as the baseline, which is the default frequency governor on many mobile devices [6]. We also compare with the best-existing governor found from mainstream CPU governors, including the `interactive` and other four strategies: `performance`, `conservative`, `ondemand` and `powersave`.

This work was partly supported by the NSFC under grant agreement 61872294 and the UK Royal Society under grant agreement IE161012.

Table I: The best-existing available governor

	Load time	Energy consumption	EDP
Regular 3G	`conservative`	`powersave`	`powersave`
Regular 4G	`ondemand`	`conservative`	`ondemand`
WiFi	`performance`	`ondemand`	`interactive`

Motivation results. Table I lists the best existing-CPU-frequency governor and Figure 1 shows the performance of each strategy for three *lower is better* metrics: load time, energy consumption and the energy delay product (`EDP`) - calculated as energy *times* delay. The widely used `interactive` governor fails to deliver the best-available performance for load time, energy consumption and `EDP`. Furthermore, running the CPU at the highest frequency (i.e., the Performance governor) is not always profitable when optimizing for load time. This is because doing so may lead to frequent CPU throttling which effectively forces the CPU to run at a lower frequency to prevent it from overheating. On average, the `oracle` outperforms the best-existing governor by 54.6%, 70.6% and 85.4% respectively for load time, energy consumption and `EDP` across networking environments. We also observe that there is no "one-fits-for-all" best CPU configuration.

This example shows that the best processor configuration depends on the network and the optimization goal, and the current mainstream CPU frequency governors are ill-suited for mobile web browsing. There is a need for a better scheduler that can adapt to the webpage workload, the networking environment and the optimization goal.

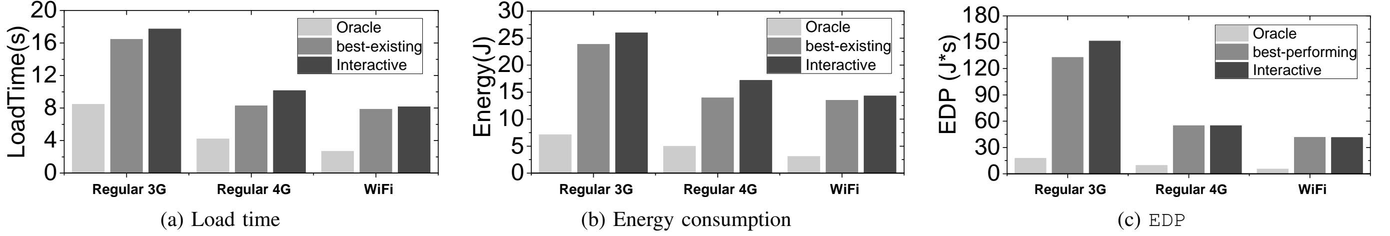
III. Our Approach

A. Overview

Our approach consists of two main components: (i) a network monitor running as an OS service and (ii) a web browser extension. The network monitor measures the end to end delay and network bandwidths when downloading the webpage. The web browser extension determines the best processor configuration depending on the network environment and the web contents. At the heart of our web browser extension is a set of *off-line* learned predictive models. The predictor takes in a set of numerical values, or *features values*, which describes the essential characteristics of the webpage and the networking environment. It predicts which core to use to run the rendering process and at what frequency the heterogeneous processors should operate. We have implemented our techniques in the open-sourced Google Chromium web browser. Note that on modern web browsers, content rendering takes place concurrently with downloading. Therefore, we want to determine the optimal processor configuration as soon as possible.

B. Predictive Modeling

Our models for processor configuration prediction are a combination of Support Vector Machines (`SVMs`) and Artificial Neural Network (`ANNs`). The `SVMs` are machine learning classifiers, each of which is specifically tuned for a typical network environment and optimization goal and is highly effective for the targeting environment. Our two `ANNs` are regression-based models: one for predicting load time, and the other for predicting energy consumption. They can be used in any environment – by taking the network latency/bandwidth and the CPU frequency setting as the model input to produce the time or energy estimations – but they could be less accurate (see Section V-F). This combination results in a generalized framework for any network environment, while provides a high confidence for typical network environments.



(a) Load time (b) Energy consumption (c) EDP

Figure 1: The achieved load time (a), energy consumption (b) and EDP (c) when browsing four webpages from news.bbc.co.uk.

Table II: Networking environment settings

	Uplink bandwidth	Downlink bandwidth	Delay (ms)
Regular 2G	50kbps	100kbps	1000
Good 2G	150kbps	250kbps	300
Regular 3G	300kbps	550kbps	500
Good 3G	1.5Mbps	5.0Mbps	100
Regular 4G	1.0Mbps	2.0Mbps	80
Good 4G	8.0Mbps	15.0Mbps	50
WiFi	15Mbps	30Mbps	5

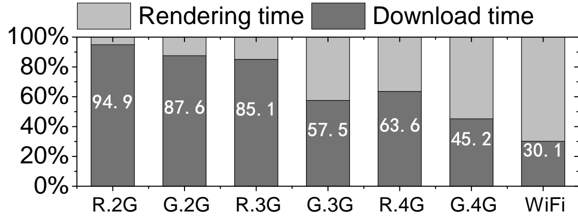
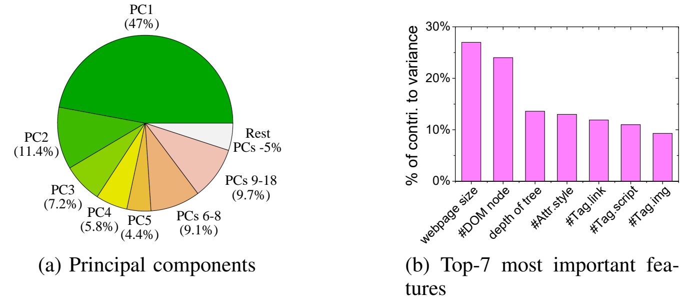


Figure 2: Webpage rendering time w.r.t. content download time when using the interactive governor.

Table III: Raw web feature categories

DOM Tree	#DOM nodes #each HTML tag	depth of tree #each HTML attr.
Other	size of the webpage (Kilobytes)	



(a) Principal components (b) Top-7 most important features

Figure 3: The percentage of principal components (PCs) to the overall feature variance (a), and contributions of the seven most important raw features in the PCA space (b).

1) Network Monitoring and Model Selection: Table II lists the networking environments considered in this work. The settings and categorizations are based on the measurements given by an independent study [7]. Figure 2 shows the webpage rendering time with respect to the download time under each networking environment when using the interactive governor. The download time dominates the turnaround time for a 2G and a Regular 3G environments; and by contrast, the rendering time accounts for most of the turnaround time in a Good 4G or WiFi environment.

We develop a lightweight network monitor to measure the bandwidths and delay between the web server and the device. Measurements are averaged over the measurement window. We calculate the distance, d, between the current network conditions and the pre-defined settings in Table II. The distance, d, is calculated as:

$$d = \alpha|db_m - db| + \beta|ub_m - ub| + \gamma|d_m - d| \quad (1)$$

where db_m, ub_m, and d_m are the measured downlink bandwidth, upload bandwidth and delay respectively, db, ub, and d are the downlink bandwidth, upload bandwidth and delay of a network category, and α, β, γ are weights. The weights are learned from the training data, with an averaged value of 0.3, 0.1 and 0.6 respectively for α, β, and γ.

If the distance to a pre-defined environment is less than a threshold, a SVM model specifically tuned for that environment and the optimization goal will be chosen. Otherwise, the two ANNs will be chosen to predict the load time and energy consumption for the current network under a given CPU frequency setting; the predictions are then used to search for a processor configuration to best satisfy the optimization constraints. We note that the distance threshold is automatically learned from training data by varying the network settings (see also Section V-F).

2) Training: Our models are built *offline* using a set of training webpages. In this work, we use 900 webpages for training and 100 unseen webpages for testing. The training webpages are selected from the landing page of the top 1,000 hottest websites ranked by alexa.com. We use Netem [8], a Linux-based network enumerator, to emulate various networking environments to generate the training data. To collect training data for SVMs, we exhaustively execute the rendering engine under different processor settings and record the optimal configuration for each optimization goal and each of the networking environments that we target. We give each optimal configuration a unique label. To collect training data for ANNs, we vary the network bandwidths and delay, as well as the processor settings for each training webpage, we then record the load time and energy consumption.

Building the model. The feature values together with the desired prediction outcomes (categorized labels for SVMs, numerical values for ANNs) are supplied to a supervised learning algorithm to learn a SVM or ANN model. Because we target three optimization metrics and seven networking environments, we have constructed 21 SVM classifiers in total. The ANNs take five extra feature values, the download/upload bandwidths, delay, and a label indicating which processor (big or little) to use and its clock frequency.

3) Web Features: We consider a set of features extracted from the document object model (DOM) tree. We started from 214 raw features (grouped into categories in Table III), including the number of DOM nodes, HTML tags and attributes of different types. Features collection takes place during runtime parsing prior to rendering. Since parsing accounts for only 1% of the total web rendering time, it leaves much room for optimization. To adapt to the change of available information, we will make a re-prediction if there are significant changes in the DOM (more than 30% of the DOM nodes), although this rarely happens in our experiments. The collected feature values are encoded to a vector of real values.

Feature reduction. To improve the generalization ability of our models, we reduce some features through applying Principal Component Analysis (PCA) [9] to the raw feature space. After applying PCA to the 214 raw features, we choose the top 18 principal components (PCs) which account for around 95% of the variance of the original feature space. We record the PCA transformation matrix and use it to transform the raw features of the new webpage to PCs during runtime deployment. Figure 3a illustrates how much feature variance that each component accounts for. It shows that predictions can accurately draw upon a subset of aggregated feature values.

Feature normalization. Before passing features to a machine learning model, we need to scale each of the features to a common range

(between 0 and 1) to prevent the range of any single feature being a factor in its importance. We record the minimum and maximum values of each feature in the training dataset, and use these to scale the corresponding features for unseen webpages in deployment.

Feature analysis. Figure 3b shows the top 7 dominant features based on their contributions to the PCs. Features like the webpage size and the number of DOM nodes are most important, because they strongly correlate to the download time and the complexity of the webpage. Other features like the depth of the DOM tree, and the numbers of different attributes and tags, are also useful, because they determine how the webpage should be presented and the rendering cost.

IV. Experimental Setup

A. Hardware and Software Platform

Evaluation platform. Our evaluation platform is an Odroid XU3 board with two heterogeneous ARM processors. The board runs Ubuntu 16.04 with the big.LITTLE enabled scheduler. We use the on board energy sensors to measure the energy consumption of the entire system. These sensors have been proven to be accurate [10]. We implemented our approach in Chromium (v64.0) which is compiled using GCC (v7.2).

Networking environments. To ensure that our results are reproducible, we use a Linux server to record and replay the server responses through the Web Page Replay tool [11]. Our mobile test board and the web server communicate through WiFi, but we use Netem [8] to control the network delay and server bandwidth to simulate the seven networking environments defined in Table II. We injected 60% of variances (normal distribution) to the bandwidths, delay and packet loss. Note that we ensure that the network variances are the same during the replay of a test page.

Workloads. We used the mobile version landing page of the top 1,000 hottest websites from alexa. The DOM node and webpage sizes of our test data range from small (4 DOM nodes and 40 KB) to large (over 8,000 DOM nodes and 6 MB), and the load time is between 0.13 and 15.4 seconds in a WiFi environment using interactive.

B. Evaluation Methodology

We use *10-fold cross-validation* to evaluate our machine learning models. We report the *geometric mean* of each metric across evaluation scenarios and cross-validation folds. We compare our approach with two alternative approaches, a web-aware scheduling mechanism (termed as WS) [4] and a machine learning based web browser scheduling scheme (termed as S-ML) [5]. We also compare our approach against five widely used CPU governors: interactive, powersave, performance, conservative, and ondemand.

V. Experimental Results

A. Compared to Existing Linux Governors

The box-plot in Figure 4 depicts the improvements of our approach over the best-performing Linux CPU governor. Figure 4a shows the improvement of load time. There is minor improvements only in a fast network like a WiFi environment. In such an environment, the download speed is no longer a bottleneck and running the CPU at a high frequency is often beneficial. Nonetheless, our approach outperforms the best performing Linux governor by 1.32x on average (up to 1.9x) across network environments and never gives worse performance. Figure 4b compares our approach against other governors in scenarios where low battery consumption is desired. In this case, powersave is the best-performing Linux governor in 2G and a Regular 3G environments, while conservative and ondemand are the best-performing policies in a faster environment (Good 3G onwards). On average, our approach outperforms the best-performing Linux governor by consuming less than 44% to 67% (up to 93%) of energy across networks. Figure 4c shows the significant improvement is available in relative slow network like a 2G and 3G environment, where our approach gives over 55% reduction on EDP.

B. Compared to Alternative Schemes

The violin diagrams in Figure 5 compares our approach against WS and S-ML and their extensions. We extended WS and S-ML to take in network bandwidths/delays as model inputs, which are respectively called WS-network-aware and S-ML-network-aware. To evaluate the performance gains by employing a more advanced machine learning technique compared to the linear model of WS, we also excluded ANNs and network features from our approach (namely, ours-network-unaware) . In this experiment, the baseline is the best-performing Linux governor.

Considering the network environments thus boosts the performance of WS and S-ML by 49.8%, 58.6%, and 59.3% respectively for load time, energy consumption and EDP. This reinforces our observation that the network conditions should not be ignored when performing web browsing optimization. The *network-aware* S-ML extension achieves the most close performance to our approach if the evaluated networking environments match the ones the SVM models are trained for. This is not supervising as both approaches employ SVMs. However, our approach is able to generalize to a wide range of environments through the use of ANNs. As a result, our approach outperforms the *network-aware* WS and S-ML by at least 32.8% (up to 49.7%) across evaluation metrics. If we take out the network features from our models (i.e., ours-network-unaware), the performance of our approach will drop by 62% albeit it still outperforms the vanilla WS (due to the non-linear modeling capability of SVMs) and S-ML (due to better model features). Considering now the improvement distribution. There are more data points at the top of the diagram under our scheme. This means our approach delivers stronger performance on more webpages than others. Overall, our approach outperforms all other schemes with an averaged improvement of 32.1%, 56.2% and 62.8% respectively for load time, energy and EDP over the baseline, and never delivers worse performance.

C. Compared to Oracle Performance

Figure 6 compares our approach with the oracle predictor, showing how close our approach is to the theoretically perfect solution. Our approach achieves 85%, 94% and 92% of the oracle performance for load time, energy consumption and EDP respectively.

D. Overhead

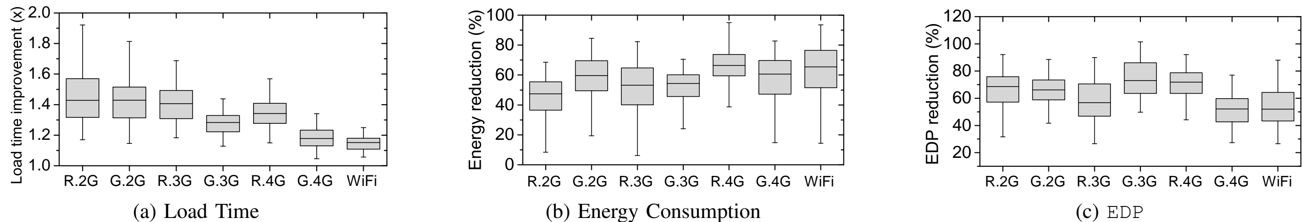
Figure 7 shows the overhead of our approach (already included in our experimental results). Our approach introduces little overhead to the turnaround time and energy consumption, less than 1% and 3% respectively. The majority of the time and energy are spent on network monitoring for measuring the network delay and bandwidths. The overhead incurred by the browser extension and the runtime scheduler, which include task migration, feature extraction, making prediction and setting processor frequencies, is less than 0.8%, with task migration (around 10ms) accounts for most of the overhead.

E. Prediction Accuracy

Our approach gives correct predictions for 85.1%, 90.1% and 91.2% of the webpages for load time, energy consumption and EDP respectively. For webpages that our approach does not give the best configuration, the resulting performance is not far from the optimal.

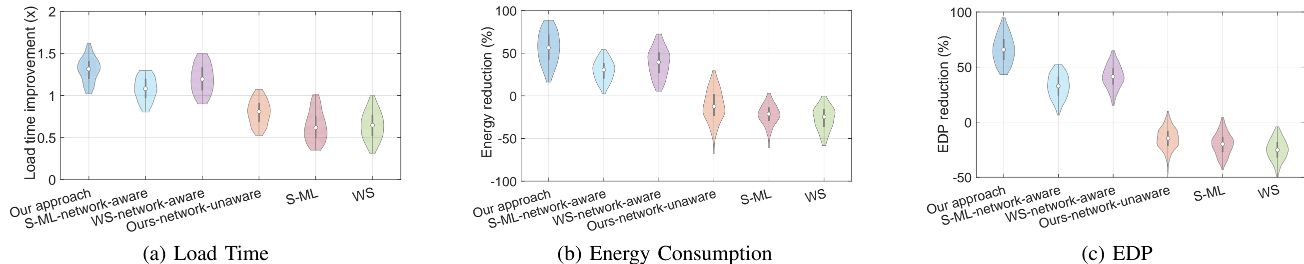
F. Impact of Model Selection Threshold

Figure 8 compares the prediction accuracy for using SVMs and ANNs alone by changing the model selection threshold, d (see Section III-B1). The percentage on the top of each bar shows how often a SVM or ANN model is chosen under a distance threshold across 1,000 different network environments. In this evaluation, we show the prediction accuracy when optimizing for load time, but we observe similar behaviors for the other two metrics. The model selection threshold, d, is calculated as the percentage to the averaged distance, l, among any of the two environments in Table II using Equation 1. The larger the threshold is, the more likely the SVM models will be chosen. For example, if d is set to 30%, it means that we only use a SVM if the distance between the current network to one of the environments listed in Table II is within 30% of l. This diagram shows that SVMs are more effective (with an accuracy of around 90%) if the current network is similar (when d is set to less than 20% of l) to the ones they are trained for, but ANNs are able to pick up the rest environments with an accuracy of at least 80%. The results show that our approach can generalize to a wide range of network environments, while providing good performance in typical networks by employing network-specific SVM models.



(a) Load Time (b) Energy Consumption (c) EDP

Figure 4: Improvement achieved by our approach over the best-performing Linux CPU governor for load time, energy reduction and EDP. The min-max bars show the range of performance improvement across webpages.



(a) Load Time (b) Energy Consumption (c) EDP

Figure 5: Distributions for load time (a), energy reduction (b) and EDP (c) in different networks. The baseline is the best-existing Linux CPU governor. The thick line shows where 50% of the data lies. The white dot is the position of the median.

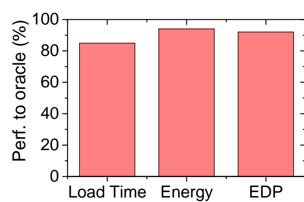


Figure 6: Our performance w.r.t. oracle.

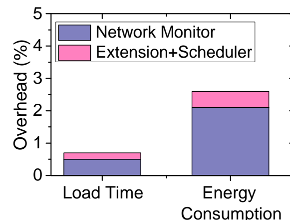


Figure 7: Breakdown of runtime overhead.

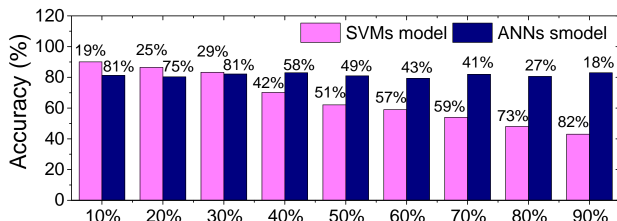


Figure 8: The prediction accuracy of using SVMs and ANNs alone with different model selection thresholds. The percentage of each bar shows how often the model is chosen across 1,000 network settings.

VI. Related Work

Techniques have been proposed to optimize web browsing, through e.g. prefetching [12] and caching [13] web contents, or re-constructing the browser workflow [14]. Most of the prior work target homogeneous systems and do not optimize across networking environments. The work presented by Zhu *et al.* [4] and our prior work [5] were among the first attempts to optimize web browsing on heterogeneous mobile systems. Both approaches use statistical learning to estimate the optimal configuration for a given web page. However, they do not consider the impact of the networking environment, thus miss massive optimization opportunities. Bui *et al.* [15] use analytical models to determine which processor core to use to run the rendering process. The drawback of using an analytical model is that the model needs to be manually re-tuned for each individual platform to achieve the best performance. Our approach avoids the pitfall by automatically learning how to best schedule rendering process. There are also works use statistical modeling or control theories to optimize energy efficiency on mobiles [16], [17], [18], [19], [20], [21]. While not specific to web browsing, these studies demonstrate the advantages and needs for adaptive system-level optimizations.

VII. Conclusions

This article has presented an automatic approach to optimize web rendering on heterogeneous mobile platforms. We achieve this by using machine learning to develop predictive models to predict which processor core to use to run the web rendering process and the optimal frequency of the processors. As a departure from prior work, our approach considers of the network status, web workloads and the optimization goals. We evaluate our approach by applying it to the Chromium browser on a big.LITTLE mobile platform using the top 1000 hottest websites. Experimental results show that our approach outperforms the state-of-the arts by 1.32x , 1.56x and 1.62x for load time, energy consumption and EDP respectively.

References

[1] S. Insights, "Mobile marketing statistics compilation," http://www.smartinsights.com/, 2016.
[2] Y. Cao *et al.*, "Deconstructing the energy consumption of the mobile page load," in *SIGMETRICS '17*.
[3] Y. Zhu *et al.*, "Event-based scheduling for energy-efficient qos (eqos) in mobile web applications," in *HPCA '15*.
[4] Y. Zhu and V. J. Reddi, "High-performance and energy-efficient mobile web browsing on big/little systems," in *HPCA '13*.
[5] J. Ren *et al.*, "Optimise web browsing on heterogeneous mobile platforms: a machine learning based approach," in *INFOCOM '17*.
[6] W. Seo *et al.*, "Big or little: A study of mobile interactive applications on an asymmetric multi-core platform," in *IISWC '15*.
[7] "State of mobile networks: Uk," https://opensignal.com/reports/, 2017.
[8] S. Hemminger *et al.*, "Network emulation with netem," in *Linux conf au*, 2005.
[9] G. H. Dunteman, *Principal components analysis*, 1989, no. 69.
[10] C. Imes and H. Hoffmann, "Bard: A unified framework for managing soft timing and power constraints," in *SAMOS '16*.
[11] (2015) Web page replay. https://goo.gl/derphf.
[12] Z. Wang *et al.*, "How far can client-only solutions go for mobile browser speed?" in *WWW '12*.
[13] F. Qian *et al.*, "Web caching on smartphones: ideal vs. reality," in *MobiSys '12*.
[14] B. Zhao *et al.*, "Energy-aware web browsing on smartphones," *IEEE TPDS*, 2015.
[15] D. H. Bui *et al.*, "Rethinking energy-performance trade-off in mobile web page loading," in *MobiCom '15*.
[16] D. C. Snowdon *et al.*, "Koala: A platform for os-level power management," in *EuroSys '09*.
[17] A. Roy *et al.*, "Energy management in mobile devices with the cinder operating system," in *EuroSys '11*.
[18] M. Kim *et al.*, "xTune: a formal methodology for cross-layer tuning of mobile embedded systems," *ACM TECS '13*.
[19] N. Mishra *et al.*, "Caloree: learning control for predictable latency and low energy," in *ASPLOS '18*.
[20] B. Taylor *et al.*, "Adaptive optimization for opencl programs on embedded heterogeneous systems," in *LCTES '17*.
[21] Z. Wang and M. O'Boyle, "Machine learning in compiler optimization," *Proceedings of the IEEE*, 2018.