Cloud-guided QoS and Energy Management for Mobile Interactive Web Applications

Wooseok Lee[1], Dam Sunwoo[2], Andreas Gerstlauer[1], and Lizy K. John[1]
[1]The University of Texas at Austin, [2]ARM Research

Abstract—**In mobile interactive web applications, energy-efficient quali-ty-of-service (QoS) scheduling involves setting a deadline for the best user experience and providing just enough performance to minimize energy. Such performance-slacking approaches require precise performance adjustment using execution time prediction. However, prior prediction approaches suffer from prohibitive training due to extensive input data and manual source code instrumentation. In this paper, we propose a cloud-guided QoS and energy management approach that eliminates the prediction overhead by offloading it to cloud resources. Our approach pre-computes per-input execution time models by profiling web applications on dedicated mobile devices in the cloud. When mobile web applications request data to servers, both the data and its execution time models are delivered to users' mobile devices. Based on the delivered models, a performance control agent on the mobile device selects an operating point to meet the response time requirement. Experimental results show that, by offloading modeling and prediction overheads, our performance-slacking approach can provide average energy savings of 22% and 39% (and up to 89%) for two different timing budgets compared to an industry-quality approach.**

I. Introduction

Mobile interactive web applications require both responsive interaction and energy efficiency. Tasks should complete within a certain deadline to prevent compromising user experience [14]. Conversely, the energy spent for improving response time should be minimal for better energy efficiency. Therefore, setting a deadline and providing just enough computing capacity is a way to minimize energy while achieving the best user experience. Various prior works [21], [7], [9], [17] have demonstrated the efficacy of such an approach.

However, the challenge in this performance-slacking approach is finding the right operating points to finish tasks just before the deadline. To determine an operating point, performance controllers such as dynamic voltage and frequency scaling (DVFS) governors need execution time models that include how much time a task takes to finish a job at each operating point. Zhu et al. [21] find the optimal operating points through online performance modeling. However, this requires multiple runs to build precise models on mobile devices. Furthermore, this approach is oblivious to input, and thus unable to capture execution time variations from distinctive input data. For instance, web pages with heavy Javascript computation show long execution time compared to ones with simple rendering.

Recent studies [9], [17] have proposed ways to improve prediction accuracy by training the execution time models with input data. Such approaches extract a program slice based on source code instrumentation [18], [19], and train it offline/online to capture program state changes originating from inputs. The program slice is executed with input data prior to the actual application run to predict the execution time. However, these training-based approaches are costly since they require manual source code instrumentation. In addition, web applications that have virtually unlimited, diverse, and ever-changing input data require constant computation for the model update.

In this paper, we present a cloud-guided QoS and energy management approach (cQoS) that eliminates the training and prediction overhead on users' mobile devices using abundant resources in the cloud. Our approach moves the modeling framework to the cloud while keeping the performance management on users' mobile devices. The modeling framework runs web applications on mobile devices in the cloud, defined as the *device farm*, where various mobile devices are ready for execution time modeling. The framework measures execution times with specific input data on various mobile devices and constructs per-input and per-device execution time models. This eliminates input- and platform-dependent execution time variations, allowing precise execution time models without prediction. Whenever a mobile device requests data, the pre-computed execution time models are delivered to the mobile device as metadata. Then, a performance control agent on a user's mobile device determines an operating point to meet the response time requirement based on the delivered models and given time budget.

We evaluate our approach on a state-of-the-art mobile heterogeneous system, a Samsung Exynos 5422 system-on-chip (SoC). Evaluation results show that, without additional model training, source code instrumentation, and prediction on users' mobile devices, our approach can successfully implement precise performance-slacking. cQoS can deliver 22% on average and up to 84% energy savings for a 500ms time budget, and 39% on average and up to 89% savings for a 800ms time budget compared to an industry-quality power management approach. This study opens the door to leveraging cloud resources for obtaining essential information required to best manage QoS and energy of mobile devices.

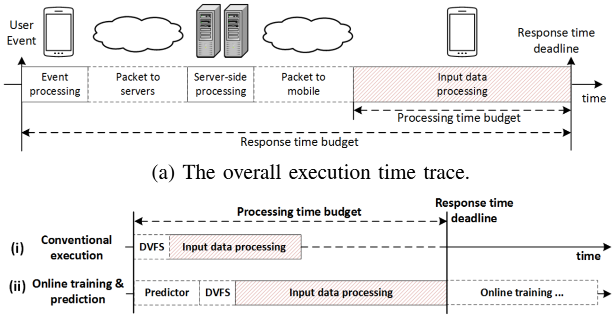
The rest of this paper is organized as follows. Section II discusses prior energy-efficient QoS scheduling approaches through execution time prediction and their limitations. In Section III, we propose our cloud-guided QoS and energy management approach. Section IV presents experimental results and we conclude in Section V.

II. Background

A. Energy-efficient QoS Scheduling

Various studies [14], [13], [20] have introduced how to quantify user experience given QoS levels (i.e., web page loading time in web browsing services). Recent studies have further proposed optimal ways to minimize energy while delivering the best user experience in interactive web applications.

Figure 1a shows the execution trace of mobile interactive web applications. When a user generates an event, the web application on the mobile device processes the event and



(a) The overall execution time trace.

(b) Input processing time trace of various approaches.

Fig. 1: Execution time trace of mobile interactive web applications.

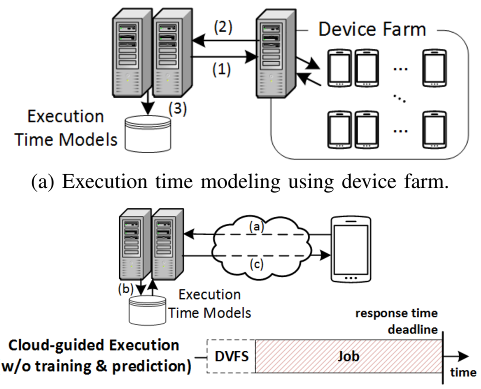
requests data through the network. The server-side application processes the request and replies back with the data. Once the data arrives, the mobile application starts computing the input data, in which most mobile-side computing energy is consumed. The computation should finish before the designated web page loading time (response time deadline in Figure 1a) to deliver the best user experience. To achieve this, the performance controller, such as the DVFS governor, should select appropriate operating points. However, this is only possible if the performance controller knows about the expected execution times at every operating points. Current mobile systems that cannot predict execution time often run applications as fast as possible in order not to violate the deadline ((i) in Figure 1b). This often leads to higher computing energy consumption which could have been saved by lowering the operating point.

B. Training-based QoS Scheduling

Recent training-based approaches [9], [17] allow execution time prediction, and thus can adjust the operating point to finish jobs in a timely manner ((ii) in Figure 1b). Using static program analyses and source code instrumentation, these approaches extract a program slice that includes key features for execution time prediction, and train the program slice with various input data. At runtime, the program slice is executed with current input for execution time prediction. The predicted execution time is used to determine the right operating point. For new input data, the post computation phase updates the execution time models in the online modeling approach [17].

While the training-based approaches provide ways to predict execution time, we find various limitations when implementing them in realistic web service environments.

- Although training with input data allows execution time prediction, web applications with virtually unlimited, distinctive, and ever-evolving input data require constant computation for the model update.
- Manual source code instrumentation to extract the program slice is prohibitive for complex web applications, such as Chrome [16] and Opera [5]. In addition, constructing models based on static analyses often fails to capture runtime interactions between threads.
- Prediction on mobile devices has negative impact on energy. Additional computation for the predictor shortens time budget for input data processing, leading to higher operating points, and thus higher energy consumption. Moreover, the energy used for the predictor and online training lowers overall energy efficiency.



(a) Execution time modeling using device farm.

(b) Cloud-guided energy-efficient QoS scheduling.

Fig. 2: Cloud-guided QoS and energy management.

III. Cloud-guided QoS and Energy Management

In this paper, we propose our cloud-guided QoS and energy management approach (cQoS) to mitigate the limitations of prior works through offloading. Unlike conventional computation offloading approaches [10], [12], [11], we do not offload computations required for applications. Instead, we propose to offload the modeling work to cloud resources. Specifically, our approach constructs per-input models using the execution time measurements of actual web application runs on mobile devices, a black-box approach that requires no source code instrumentation and training. We detail the modeling procedures and how mobile devices can take advantage of the information to manage QoS and energy in the following sections.

A. Overview

First, our approach constructs the execution time models for a specific device to eliminate the errors from targeting various computing platforms. For the per-device modeling, we depend on cloud resources, defined as the *device farm*, where various mobile devices are available for the modeling work. Although there might be some limitations in placing all types of mobile devices in the device farm, we assume that all user mobile devices can be represented by some devices in the farm. In addition, since the device farm constantly supplies power to the mobile devices, our approach does not account for the energy consumed for the modeling work.

Our approach also constructs input-specific models by running applications with specific input data and measuring execution times. Using the actual measurements, we can not only eliminate heavy computation for training, but also deliver precise execution time estimation since they are specific to the input data. Moreover, measuring execution time allows us to naturally capture the inherent complex interactions of multiple threads in web applications.

However, even with the actual runs, the execution time variations from micro-architecture [7], scheduling, runtime interferences, etc., exist. We capture these variations using statistical modeling. In addition, the execution times of all operating points are modeled in order to quickly select one operating point without additional computation at runtime. We assume that multiple mobile devices in the *device farm* can parallelize these procedures.

Figure 2a shows the overall modeling procedure. Whenever input data such as a web page is initially created or updated,

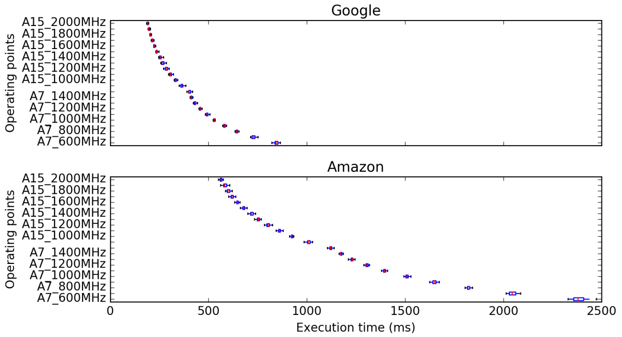


Fig. 3: Example statistical execution time models.

the server (1) sends a modeling request to the device farm with the input data. The host computer in the device farm deploys the data to the mobile devices to execute and collect execution times. The results are transferred to the host computer and used for model construction. The mobile devices load the input data from file systems in order to eliminate network latency. The execution time models are (2) returned back to the server and (3) stored in the database for future use.

Figure 2b shows the overview of our cloud-guided QoS scheduling. (a) A mobile device sends a data request to the server with the information about the requesting mobile device. (b) The server searches for the execution time models associated with the data and the mobile device in the database. (c) Both the input data and its models are delivered to the mobile devices, and the performance controller selects the minimum operating point that can finish the task before given time budget without any pre- or post-computation for prediction and online training, respectively.

B. Cloud-based Execution Time Modeling

Our target system is a recent mobile SoC, Samsung Exynos 5422, which includes quad Cortex-A15 (2GHz-800MHz) and quad Cortex-A7 (1400MHz-600MHz) processors. BBench [6] is used as workloads for the web page loading tasks. The detailed experimental setup will be discussed in Section IV.

As discussed in the previous section, we capture the execution time variations using statistical modeling. Specifically, the host computer in the device farm computes mean and standard deviation values from the measured execution times. For statistical guarantee, it repeats the executions 50 times. Figure 3 shows the two execution time models for Google and Amazon web pages. Since the Google page contains negligible computation, the web page loading takes a short amount of time (189ms-845ms). However, since the Amazon page includes multiple objects that require more computation and rendering, it takes a relatively long time to load (562ms-2387ms). To avoid deadline misses, the 95 percentile execution time (mean+2σ) is used as the decision making point. The models can be represented in the form of Table I. Since the table size can be as small as several bytes, we believe that there is negligible overhead for storing and transmitting it.

TABLE I: Example execution time models when loading Google web page on Samsung Exynos5422 SoC.

	Execution Time (ms)	
Operating Point	Mean	Std.
2000MHz (A15)	189	2
...	...	...
1400MHz (A7)	412	7
...	...	...

Algorithm 1 Performance Control Agent

```
$time\_budget$ = qos_time - time_elapsed        ▷ time in ms
**if** $time\_budget < 0$ **then**
    $freq \leftarrow$ highest freq
    $core \leftarrow$ performance core
**else**        ▷ find min opp s.t. $(mean + 2\sigma < time\_budget)$
    $freq, core \leftarrow$ find_min_opp($time\_budget$)
**end if**
set_freq($freq$)
set_coretype($core$)
```

C. Cloud-guided QoS Scheduling

In mobile devices, a performance control agent determines the operating point based on the delivered models given the time budget left for input data processing. Algorithm 1 shows the detailed procedures of the performance control agent. When the agent receives the execution time models, it first evaluates the time budget. If there is no time budget, it selects the highest operating point to minimize the response time. Since our target system has heterogeneous core types, it selects the highest frequency of the high-performance core (Cortex-A15). If any time budget is left, the agent finds a minimum operating point in the models that meets the budget. Then, the agent interacts with the DVFS governor and scheduler to run the web applications on the designated frequency and core type. In this paper, we use the *userspace* governor for the frequency change and the *cgroup* for the core type selection. Evaluation results show that the worst-case DVFS switching time is 2ms and the time for selecting core types is negligible.

The performance control agent has the responsibility of managing QoS and energy as one of the subtasks in web browsers. When a user generates an event, the agent sends the mobile device information to the target server, keeps track of time for the event, and makes an operating point selection decision. In this paper, for the prototyping of our approach, we create the external agent threads both in the server and the mobile device and use sideband channels to communicate between the agent and server, although it may be possible to embed the data communication in standard protocols in the future. The server thread holds the pre-computed models and sends the execution time models when the agent thread requests the data. The agent thread constantly monitors user events by reading input events in the Linux kernel for stepping in at the user event and changes the operating point after the arrival of the execution time models.

IV. Experiments

Experimental setup: For the evaluation of our approach, we use ARM's big.LITTLE platform which has been commonly employed in many mobile systems. In particular, we select an ODROID-XU3 board [4] that includes Exynos 5422 SoC containing quad Cortex-A15 and quad Cortex-A7 cores. Energy is collected from on-board energy sensors, and, for precise energy collection, we reduce the sampling period to 1 ms by modifying kernel drivers. ARM's DS-5 Streamline [2] is used to deliver the sampled data from the board to the host machine through an Ethernet connection. We repeat the experiments 10

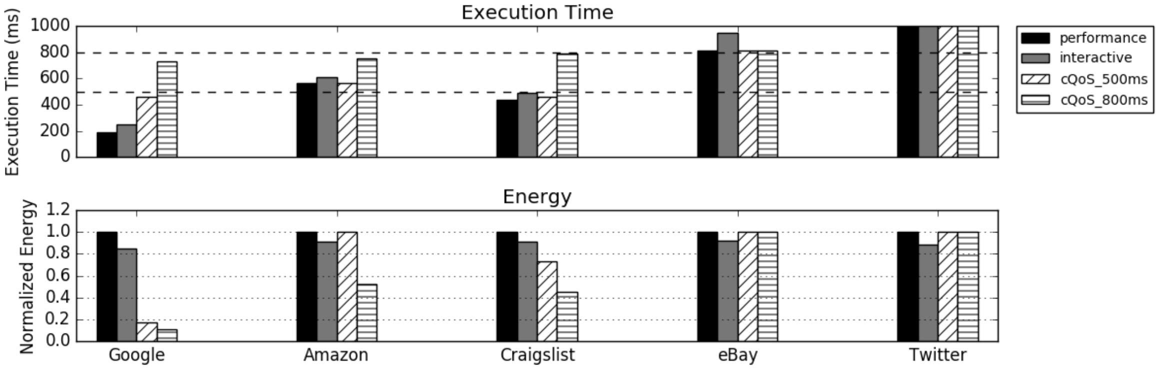


Fig. 4: Execution time and normalized energy when loading various web pages.

times and report average values for web page loading time and energy on a Chrome browser.

Industry-quality approaches: We compare our evaluation results with two industry-quality QoS and energy management approaches.

- Performance approach: Tasks in this approach run at the highest operating points to minimize response time. In our target platform, web applications run on quad-core Cortex-A15 at 2GHz; using only quad high-performance cores has no or negligible performance impact [8].
- Interactive approach: This approach follows industry-quality performance and energy management using *interactive* governor with the heterogeneity-aware scheduler [1], [3].

Time budget: Although various studies [14], [21], [7] have similar definitions of QoS, the QoS requirements, e.g., the response time budget, are defined in multiple ways. In this paper, we define the response time requirement for web page loading as 1 second, meaning that user experience does not degrade if web page finishes loading within 1 second. However, the actual time budget left for the input data processing differs mainly due to the network latency. To observe the energy impact from the varying time budget, we evaluate two types of network delays: 500ms for a slow network and 200ms for a fast network, leaving the input processing time budget at 500ms and 800ms for slow and fast networks, respectively.

Workloads: Although real computations are different, we find similarities in performance and energy trend among workloads in BBench. Therefore, we select several web pages (Google, Amazon, Craigslist, eBay, and Twitter) that show distinctive characteristics for our experiments.

Experimental results: Figure 4 shows the execution time and energy of various workloads. Since the performance approach runs at the highest operating points, it delivers the shortest execution time, and thus, consumes the highest energy. The interactive approach has similar operating point selections with the performance approach, but shows slightly lower performance due to the initial response delay, and thus, lower energy. However, this over-provisioned performance for the fast response time may provide no practical benefit to user experience.

For the light computing workloads such as Google, web page loading finishes as early as 200ms, leaving huge room for energy savings. In the performance and interactive approaches, computation is mostly done at the highest operating point (2GHz on A15) while some minor tasks in the interactive approach run on lower operating points (e.g., on A7). This allows the interactive approach to save 15% energy compared to the performance approach at the cost of slight performance degradation. Our approach can select much lower operating points due to the reduced computing requirement (about 300ms input processing time budget). This allows computation in lower operating points, mostly on energy-efficient cores (A7), resulting in huge energy savings, about 83% and 89% for 500ms (denoted as cQoS_500ms) and 800ms (denoted as cQoS_800ms) time budget, respectively, compared to the performance approach.

The medium computing workloads, Amazon and Craigslist, can save energy about 48% and 55%, respectively, only for a 800ms time budget since their loading time is about 500ms, leading to no margin for performance-slacking for a 500ms time budget. The heavy computing workloads, eBay and Twitter, have no extra margin for energy savings since the minimum computation time exceeds 800ms.

Evaluation results show that our approach can deliver energy savings averaging 22% and up to 84% for a 500ms time budget, and averaging 38% and up to 89% for a 800ms time budget, compared to the performance approach. In general, it is true that the extra time budget for input data processing leads to energy savings if it is possible to predict execution time without any computation overhead. Considering the fact that the performance of mobile processors has been continuing to increase [15], the execution time for the same workload can further decrease. This shortened execution time increases the time budget for input processing, leading to more energy savings in the future.

V. Conclusions

In this paper, we propose a cloud-guided QoS and energy management approach that eliminates training overhead and prediction cost on mobile devices by offloading execution time modeling to cloud resources. Preliminary results show that our approach can deliver energy savings of 22% and 39% on average depending on the timing budget and up to 89% compared to an industry-quality approach while requiring no overhead on mobile devices. This approach paves the way to reach out to cloud resources for mobile devices to use advanced energy management approaches without additional cost.

Acknowledgements

This work is supported by a Samsung PhD Fellowship and NSF grant CCF-1337393.

References

[1] ARM big-LITTLE Global Task Scheduler. http://www.arm.com/files/pdf/big_LITTLE_technology_moves_towards_fully_heterogeneous_Global_Task˙Scheduling.pdf.

[2] ARM DS-5 Development Studio. https://developer.arm.com/products/software-development-tools/ds-5-development-studio.

[3] Odroid kernel 3.10 git repository. https://github.com/hardkernel/linux/tree/odroidxu3-3.10.y.

[4] ODROID XU3 Development Board. http://www.hardkernel.com/main/products/prdt_info.php?g_code=g140448267127.

[5] Opera Browser. http://www.opera.com.

[6] A. Gutierrez et al. Full-System Analysis and Characterization of Interactive Smartphone Applications. In *IISWC*, 2011.

[7] B. Gaudette et al. Improving smartphone user experience by balancing performance and energy with probabilistic QoS guarantee. In *HPCA*, 2016.

[8] C. Gao et al. A Study of Mobile Device Utilization. In *In ISPASS*, 2015.

[9] D. Lo et al. Prediction-guided Performance-energy Trade-off for Interactive Applications. In *MICRO*, 2015.

[10] E. Marinelli. Hyrax: cloud computing on mobile devices using MapReduce. Technical report, 2009.

[11] H. B. et al. Mobile computing-a green computing resource. In *WCNC*, 2013.

[12] K. Kumar et al. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 2010.

[13] K. Yan et al. Characterizing, Modeling, and Improving the QoE of Mobile Devices with Low Battery Level. In *MICRO*, 2015.

[14] M. Fiedler et al. A Generic Quantitative Relationship Between Quality of Experience and Quality of Service. *IEEE Network Magazine of Global Internetwoking*, 2010.

[15] M. Halpern et al. Mobile CPU's Rise to Power: Quantifying the Impact of Generational Mobile CPU Design Trends on Performance, Energy, and User Satisfaction. In *HPCA*, 2016.

[16] N. Peters et al. Web Browser Workload Characterization for Power Management on HMP Platforms. In *CODES*, 2016.

[17] T. Song et al. Prediction-Guided Performance-Energy Trade-off with Continuous Run-Time Adaptation. In *ISLPED*, 2016.

[18] M. Weiser. Program slicing. In *ICSE*, 1981.

[19] Y. Kwon et al. Mantis: Automatic performance prediction for smartphone applications. In *ATC*, 2013.

[20] Y. Ou et al. Q-STAR: a perceptual video quality model considering impact of spatial, temporal, and amplitude resolutions. In *IEEE Trans Image Process*, 2014.

[21] Y. Zhu et al. Event-based scheduling for energy-efficient QoS (eQoS) in mobile Web applications. In *HPCA*, 2015.