HKU CS Tech Report TR-2014-06

Postprint of article in *Proceedings of the IEEE 38th Annual Computer Software and Applications Conference* (*COMPSAC '14*), IEEE Computer Society, 2014

CrowdAdaptor: a CrowdSourcing Approach Toward Adaptive Energy-Efficient Configurations of Virtual Machines Hosting Mobile Applications

Edward Y.Y. Kan
Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong
yyekan@cs.hku.hk

W.K. Chan
Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Hong Kong
wkchan@cityu.edu.hk

T.H. Tse
Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong
thtse@cs.hku.hk

***Abstract*—Applications written by end-user programmers are hardly energy-optimized by these programmers. The end users of such applications thus suffer significant energy issues. In this paper, we propose CrowdAdaptor, a novel approach toward locating energy-efficient configurations to execute the applications hosted in virtual machines on handheld devices. CrowdAdaptor innovatively makes use of the development artifacts (test cases) and the very large installation base of the same application to distribute the test executions and performance data collection of the whole test suites against many different virtual machine configurations among these installation bases. It synthesizes these data, continuously discovers better energy-efficient configurations, and makes them available to all the installations of the same applications. We report a multi-subject case study on the ability of the framework to discover energy-efficient configurations in three power models. The results show that CrowdAdaptor can achieve up to 50% of energy savings based on a conservative linear power model.**

Keywords*—*Mobile energy consumption; Test harness; Post-deployment validation; Energy optimization; Energy saving

I. Introduction

Energy efficiency is a design concern when developing software applications. It is especially important for applications running on handheld devices such as smartphones [7]. A primary reason is that using a more energy-efficient version of an application helps a device last for a longer period without the need to recharge the battery. However, applications are no longer confined to be developed by device vendors and a small list of their authorized software partners. For instance, an ordinary person may follow the procedure presented in an online video to write an original application and post the latter to an online repository (such as the Apple App Store, various Android markets, or the Windows Marketplace). End users may then install instances of the application on their handheld devices, virtual machines, personal computers, and so on. For ease of presentation, we refer to such an ordinary person as an *end-user programmer*.

End-user computing is emerging, especially in developing macro-based spreadsheet applications [3]. However, end-user programmers normally do not have the technical skills and tools to optimize their applications with respect to energy efficiency for a wide range of device models or even for a specific model. Indeed, there are a huge number of device models, some of which are only available after the applications have been created, resulting in suboptimal energy optimizations (if any). End-user programmers also do not have the resources to adapt the application to fit the most energy-efficient option of every available model. As a result, these applications are often posted in confounding forms (that is, without a "best-fit" target of energy optimization) before end users install and execute them. Hence, the burden of making an application instance operate energy-efficiently lies on individual end users, who nonetheless also do not have the skills to optimize these application instances even if they would like to do so. To the best of our knowledge, there is no solution proposed in the literature to address this challenge.

Applications written by end-user programmers may or may not pose security threats to the devices. One way to confine security threats is to execute each application instance in a standalone virtual machine (VM) instance [1][5]. We observe that this class of VM-based solution further allows each application to personalize the virtual machine that hosts the application instance.

In this paper, we exploit the above observation and the availability of multiple devices, each of which is installed with an instance of the same application to address the above challenge. We propose CrowdAdaptor, a novel framework to gradually and collectively adapt the energy-aware configurations of mobile applications toward increasingly more energy-efficient configurations for execution in standalone VM instances on handheld devices.

Specifically, an application instance instrumented with the CrowdAdaptor framework will be sampled by the framework to execute certain test cases under specific VM configurations. We exploit the presence of many devices that are installed with the same application instrumented with the CrowdAdaptor framework, to keep the sampling rate of each application instance to an extremely low level to reduce the impact on the device to the minimal. CrowdAdaptor collects a record of profiling data of a test case executed under a specific VM configuration from each sample, and collects many such records for many test cases associating with many VM configurations generated from many devices. Based on the dataset, CrowdAdaptor computes an optimal configuration for each kind of VM to host an instance of the application. Moreover, when the application instance hosted in a VM on a device is not used for providing profiling data, the instance may query CrowdAdaptor to obtain an energy-efficient option discovered up to that moment for the VM so that the application instance can gradually save increasingly more energy when executed in the native mode.

We have evaluated CrowdAdaptor in a two-subject case study, in which the entire projects of two popular real-world applications K-9 Mail [10] and MyTracks [15] (including their test cases) are used as subjects. We propose and apply three individual algorithms to select test cases and VM configurations for individual application instances to execute. The case study results show that the two subjects can successfully and gradually adapt to use more energy efficient VM options with an estimated energy saving of 0 to 50% in terms of processor energy consumption after CrowdAdaptor has been enabled in the first 100 test suite executions against each subject using the conservative *linear* power model [26]. If more advanced power models[1] are used, our results show that the saving can be even more significant: a saving of 27 to 69% (with one exceptional case) when using the *quadratic* power model, and a saving of 42 to 94% (with one exceptional case) on the *cubic* power model. Our results also show that for 4 out of 6 subject and power model combinations in the case study, the algorithms can locate the configuration with the maximum energy savings from all the available configurations.

This paper makes the following contributions: (i) To the best of our knowledge, it presents the *first* technique that uses development artifacts (namely, test cases) in crowdsourcing scenarios to address the energy-efficient optimization problem encountered by mobile applications. (ii) We report the first case study to show the effect of different search strategies in locating energy-efficient settings for two real-world mobile applications.

The rest of the paper is organized as follows: In Section II, we present the CrowdAdaptor framework. In Section III, we present the research questions to be studied in the case study, followed by the case study and its data analyses in Sections IV and V, respectively. We review closely related work in Section VI and conclude the paper in Section VII.

II. The CrowdAdaptor Framework

An application running on a virtual machine can take advantage of the CrowdAdaptor framework by operating in one of the two modes. First, in the native mode of operation, an application instance may request for the best configuration discovered by CrowdAdaptor so far for that specific type of virtual machine. Second, in the testing mode, the application instance will execute a specific test case against a specific configuration informed by CrowdAdaptor. The participating application instance provides CrowdAdaptor with performance data to support the discovery of the best configuration.

A. Application and Application Instances

A device D_z executes a virtual machine instance of type VM_i, which hosts an instance of a mobile application P developed by end-user programmers. VM_i can be specified in terms of one of the elements in a set of configuration values denoted by $\Gamma = \{C_1, C_2, \ldots, C_n\}$. A *configuration value* C_k is a collection of settings needed to define VM_i. For instance, C_k may include (a) the processor frequency subinterval (which can be set to any range of *available* clock frequencies, such as a subinterval of (384, 486, 594) MHz within an interval of (384, 486, 594, 702, 810) MHz), (b) the size of the memory allocated to the VM instance, and (c) the type of WiFi connection and its power state used (such as 802.11ad using a low power state). For the sake of brevity and better readability, we will refer to a configuration value simply as a *configuration*. We assume that one of the configurations in Γ is a default configuration independent of the CrowdAdaptor framework.

The mobile application P is designed to accept an input x from its input domain X. Under an instance of VM_i configured by C_k, the execution of x against an instance j of P generates a performance vector, which is profiled and denoted by $P_j(x, VM_i, C_k)$. There are natural variations in performance, such as the execution time and the number of bytes sent or received via VM_i configured with C_k (e.g., a wireless network configuration of the VM). Hence, different executions of the same mobile application P may produce different profiled vectors.

Without loss of generality, we define that a larger value in an entry of a performance vector indicates a more inferior performance for the entry. For instance, a value of two seconds of execution time is worse than a value of one second.

B. Device Sampling and Test Case Sampling

We recall that a test case is an input of the application P, and a set of test cases constitute a test suite T. Since the same application P can be installed on multiple devices, we model the crowdsourcing scenario that there is a set Φ of such devices that are online at any moment, and each device in Φ has been marked by end users to allow the executions of test cases against the application P. The history H of Crowd-

[1] A real-world power model of a physical processor is in between the quadratic and cubic power models.

CrowdAdaptor, initially empty, is a sequence of performance vectors $\langle P_{j1}(x_1, VM_{i1}, C_{k1}), P_{j1}(x_2, VM_{i2}, C_{k2}), ...\rangle$.

The workflow of CrowdAdaptor starts with selecting a device to execute a test case t in T. CrowdAdaptor for the application P accepts a sampling rate of r% when it initializes. When a device D_z is made online, in the sense of a heartbeat message [2][9], the device periodically registers itself to Φ, where the registration is associated with a virtual machine type VM_i. Each registration triggers CrowdAdaptor to determine with a probability of r% that in the current heartbeat period of device D_z, the device will be used by CrowdAdaptor for test execution. If the device is selected, CrowdAdaptor further determines the set $\Gamma$of configurations for VM_i and uses its optimization engine (to be presented in Section II.D) to select one configuration C_k.

Algorithm 1 specifies the high-level interaction between CrowdAdaptor and a selected device.

Algorithm 1 (Test case and configuration selection in CrowdAdaptor)

```
1:  function selectTestCaseConfig(mode, VM_i) {
2:    if mode = native {
3:      C_k ← best configuration in Γ for VM_i
4:      if C_k = ∅ { C_k ← default configuration in Γ }
5:      return ⟨C_k⟩
6:    } else if mode = testing {
7:      C_k ← runOptEngine(Γ for VM_i)
8:      t ← randomly selected test case from T′ ⊆ T
9:      return ⟨C_k, t⟩ }}
```

For devices using the native mode, CrowdAdaptor simply returns the best configuration discovered so far for VM_i at line 3. If no configuration has been discovered yet, the default configuration is used (line 4). For devices using the testing mode, CrowdAdaptor randomly selects one test case t in a subset T' of T (line 8), such that none of the test cases in T' has received any performance data regarding virtual machine type VM_i with configuration C_k since the last reset of T'. Specifically, the subset T' for VM_i with configuration C_k is initialized as T, and whenever a selected test case t generates a performance vector, the test case t is removed from T'. If T' becomes empty after the removal, then T' is reset to T. Note that CrowdAdaptor does not assume that the device must eventually return a performance vector. Hence, the test suite T' is reduced only after the performance vector is received.

After CrowdAdaptor has selected a test case t, it requests device D_z to **both** (1) execute t against the instance j of P hosted on the device's virtual machine instance of type VM_i with configuration C_k (line 9 of Algorithm 1), **and** (2) return the performance vector $P_j(t, VM_i, C_k)$. It updates the history H by appending $P_j(t, VM_i, C_k)$ to H (that is, to construct H^$\langle P_j(t, VM_i, C_k)\rangle$) whenever $P_j(t, VM_i, C_k)$ is received. If the returned vector $P_j(t, VM_i, C_k)$ triggers a reset of the test suite T', the vector is also annotated with a reset marker.

In the above design, any reset of the test suite T' with respect to virtual machine type VM_i with configuration C_k indicates that every test case has been executed at least once against a virtual machine instance of type VM_i with configuration C_k. Thus, during the reset operation, CrowdAdaptor also extracts those performance vectors that belong to VM_i configured with C_k, right after the last reset and up to the current reset, to construct a dataset PD of performance vectors. It uses PD to conduct an energy consumption assessment, which will be presented in the next subsection.

C. Energy Consumption Assessment

Some research work has been proposed to compute (or estimate) the energy consumptions based on a performance dataset [12][14][26]. Energy consumption may be computed from performance statistics based on an energy model. In the case of the processor, for instance, the relationship between the operating voltage or frequency of the processor and its energy consumption may range from linear [26], to quadratic [4][24], to cubic [12][22]. It is also possible for a device manufacturer to provide customized power profiles that directly correlate power states to energy consumption [21].

CrowdAdaptor does not invent its own proposal in this aspect. Rather, it builds on top of existing work. To preserve generality, we model an energy consumption estimation technique as a couple $\langle PV, g\rangle$, where PV is the vector of performance metrics needed by the existing technique g.

We recall we have presented in Section II.A that CrowdAdaptor profiles performance vectors. Specifically, CrowdAdaptor accepts such a couple $\langle PV, g\rangle$ as input to compute energy consumption with respect to PV.

We have presented in Section II.B that each reset of a test suite triggers CrowdAdaptor to generate a dataset PD with respect to a virtual machine type VM_i with configuration C_k. If there are multiple vectors in PD such that each is associated with the same test case, CrowdAdaptor computes a corresponding mean performance vector from the former vectors, drop all the former vectors and add the newly computed vector to PD. As such, each test case in T is associated with one vector in PD.

CrowdAdaptor then applies g to each vector in PD to compute the corresponding energy consumption values, and accumulate these consumption values to give one total value v_1. Finally, CrowdAdaptor associates v_1 with virtual machines of type VM_i with configuration C_k, represented by the triple $\langle VM_i, C_k, v_1\rangle$. To mask out natural variations in performance metrics, CrowdAdaptor collects multiple instances of such triples associated with virtual machines of type VM_i with configuration C_k. It then computes the mean of these consumption values, and uses the mean value as an energy consumption indicator E_k for this virtual machine type and configuration.

D. Optimization Engine

We have presented in the last subsection how CrowdAdaptor associates an energy consumption indicator E_k with virtual machine type VM_i having configuration C_k. In this section, we present how CrowdAdaptor finds a configuration C_k needed to configure a VM instance running on a device. In general, a configuration specifies a range of feasible values for each option, and such a range can be reduced into a constant. We first use the *policy governors* for processor frequency control in Android to illustrate the context of the problem.

For ease of reader understanding, in the sequel, we will use frequency control to present our strategies. Nonetheless, we note that these strategies are generally applicable beyond frequency control.

1) ondemand CPU Frequency Policy Governor

Each policy governor implements its own algorithm to determine which processor frequency should be used based on historical processor utilization and other factors such as user interface activity. An app with *root* access can configure the behavior of a policy governor by passing parameters to the governor.

The *ondemand* CPU frequency policy governor [18] is the de facto policy governor used by many Android device manufacturers as the default governor. It considers a subinterval *Freq'* on the sequence of discrete processor frequencies (denoted by *Freq*) supported by the VM type. Initially, the subinterval *Freq'* spans from the minimum to the maximum frequency in *Freq*. *ondemand* increases the current processor frequency *f* to the maximum frequency in *Freq'* if the processor utilization exceeds a predefined threshold kept by the variable *up_threshold*. On the other hand, if the processor utilization drops below a predefined threshold kept by the variable *down_threshold*, it decreases *f* by 20%, rounded to the nearest frequency in *Freq'*. By writing to the special operating system files *sampling_rate_min* and *sampling_rate_max*, respectively, one can control the subintervals of processor frequencies used by a VM instance.

2) Naïve Search-Based Optimization Strategy

Consider, for example, a virtual machine type such that all the available processor frequencies f_i are given by $Freq = (f_1, f_2, f_3, f_4)$, where $f_1 < f_2 < f_3 < f_4$. Fig. 1 shows a directed graph of all the subintervals of *Freq* acceptable to *ondemand*. In general, the number of nodes in the directed graph is exactly $|Freq|\ (|Freq| + 1) / 2$, which means that the search space is in the order of $O(|Freq|^2)$. For VM types supporting many frequencies, an exhaustive search of this type of graph may require too many executions of the same test suite even if there are many devices available to run the test cases. Thus, a naïve search-based optimization strategy is not quite a solution. In the next three subsections, we formulate two improved search-based strategies and a simple strategy to address this problem.

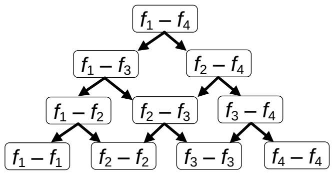


Fig. 1. A virtual processor with possible frequencies $Freq = (f_1, f_2, f_3, f_4)$.

3) Single-step Top-Down Search (STDS) Strategy

Algorithm 2 shows the *Single-step Top-Down Search* (*STDS*) optimization engine (called by line 7 of Algorithm 1) for a specific virtual machine type that supports the sequence of frequencies *Freq*. The operations on C_k can be adapted for other factors of optimization in addition to processor frequency control.

At line 2, *STDS* retrieves the best configuration discovered so far for VM_i. Initially, in lines 3–4, it asks all the devices registered to Φ to execute the test cases in *T'* using the default configuration (which is the root node f_1–f_4 in Fig. 1 in our example). The algorithm then considers the next set of configurations by updating the from/to frequency indices one step at a time (lines 15 & 17). Referring to Fig. 1, the next set of configurations are the two lower-level nodes.

During the search, devices registered to Φ will be asked to supply performance vectors for different configurations (lines 6 and 8) in order to construct the performance dataset *PD*. When sufficient performance data have been collected for the entire test suite and a more energy-efficient configuration is found, *STDS* updates the running best configuration (lines 11–12) for other devices executing in native mode.

The search terminates when either the frequency indices converge (line 4), or if a better configuration cannot be found (line 9). The algorithm has been designed with a margin of improvement to prevent the strategy from being trapped in local minima. This margin threshold may vary according to the VM type. (In our experiment, we find that 10% is a good indicator as a tunable resource for processors.)

Once the algorithm ends, the search for the best configuration (frequency subinterval in this case) is complete. In this way, a set of configurations for the same test suite has been profiled with energy consumption values, and the best configuration for the specific VM type is determined. All subsequent executions of the same application *P* on the profiled VM type can use this best-found configuration.

Algorithm 2 (*STDS* strategy to search for a VM configuration).
```
 1:  function runOptEngine(Γ) {
 2:    C_k ← best configuration in Γ
 3:    if C_k = ∅ { C_k ← {from ↦ 1, to ↦ |Freq|} }
 4:    if C_k.from = C_k.to or E_k = ∅ { return C_k }
 5:    C_left ← updateTo(C_k)
 6:    if E_left = ∅ { return C_left }
 7:    C_right ← updateFrom(C_k)
 8:    if E_right = ∅ { return C_right }
 9:    if E_k is smaller than E_left and E_right by some margin {
10:      return C_k }
11:    if E_left < E_right { best configuration in Γ ← C_left }
12:    else { best configuration in Γ ← C_right }
13:    return best configuration in Γ }
14:  function updateFrom(C_k) {
15:    return {from ↦ C_k.from + 1, to ↦ C_k} }
16:  function updateTo(C_k) {
17:    return {from ↦ C_k.from, to ↦ C_k.to –1} }
```

Algorithm 3 (*BTDS* strategy to search for a VM configuration)
```
1–13:  (same as STDS)
14:    function updateFrom(C_k) {
15:      return {from ↦ (C_k.from + C_k.to)/2, to ↦ C_k.to} }
16:    function updateTo(C_k) {
17:      return {from ↦ C_k.from, to ↦ (C_k.from + C_k.to)/2} }
```

4) Binary Top-Down Search (BTDS) Strategy

Algorithm 3 shows the algorithm for the *Binary Top-Down Search* (*BTDS*) strategy, which is identical to *STDS* except for how it searches for the next set of configurations (lines 14–17). Instead of reducing the subinterval by one step, *BTDS* shrinks it by half. The idea is to try the subintervals that are further away from the current subinterval so that more diverse subintervals are tried with fewer test executions.

5) Incremental Random Search (IRS)

We further propose a randomizing strategy called *Incremental Random Search* (*IRS*), as shown in Algorithm 4. It keeps track of a randomly selected configuration C_{irs}, and asks participating devices to execute test cases against this configuration until the energy consumption indicator is computed (line 9). If the indicator shows improvement over the running best configuration, the latter is updated (line 11), and C_{irs} is reset to another configuration in Γ*'* that has no *PD* (lines 3–6). The algorithm also keeps track of the number of configurations already tried (using the variable *trials* at line 7), and accepts a parameter *max* that limits the number of configurations to be tried across multiple calls to *runOptEngine* (line 5). The variable *trials* is initialized to zero when at the beginning of the process of finding maximum energy savings.

Algorithm 4 (*IRS* strategy to search for a VM configuration)

```
1:     // C_irs is set to ∅ and trials is set to 0 at the beginning of the
         process of finding maximum energy saving.
2:     function runOptEngine(Γ, max) {
3:       if C_irs = ∅ {
4:           Γ′ ← {C_i ∈ Γ | E_i = ∅}
5:           if Γ′ = ∅ or trials ≥ max { return best configuration in Γ }
6:           C_irs ← randomly selected configuration from Γ′
7:           trials ← trials + 1
8:           return C_irs
         }
9:       if E_irs = ∅ { return C_irs }
10:      C_k ← best configuration in Γ
11:      if C_k = ∅ or E_irs < E_k { best configuration in Γ ← C_irs }
12:      C_irs ← ∅
13:      return best configuration in Γ }
```

III. Research Questions

Through the case study to be presented in the next section, we would like to ask the following research questions in the context of energy savings in processors.

RQ1: To what extent can the application adaptively save energy through CrowdAdaptor with the most conservative (linear) power model, compared with the default configuration of *ondemand* using the entire frequency interval?

RQ2: What effect does the use of different processor power models have on CrowdAdaptor?

The two research questions evaluate, with respect to the default parameter setting, the three proposed strategies that leverage crowd-based effort to search for a best configuration.

IV. A Multi-Subject Case Study

A. Review of Processor Power Model

Dynamic Frequency and Voltage Scaling (DVFS) [24] is a feature present in most modern mobile and desktop processors that allow program control of their operating frequency. When the operating frequency is lowered, the operating voltage may also be lowered accordingly in order to achieve energy savings. Suppose *Freq* is a sequence of clock frequencies supported by a processor. Using a similar processor power model as in Xu et al. [25] and Zhang et al. [26], the computation energy consumed by a processor (in an active state) operating at clock frequency *f* can be modeled as follows:

$$t\,(\beta_f\, u + \beta_c),\ f \in Freq \qquad (1)$$

where t is the computation time, β_f is a frequency-dependent power coefficient, u is the processor utilization resulting from the execution during time t, and β_c represents the power difference between the active and idle states of the processor. The varying beliefs in the relationship between energy consumption and f as stated in the previous section, namely, linear, quadratic, and cubic, can be captured by the value of β_f with respect to *f*. Owing to page limit, for brevity, we refer the readers about the three models to the work of Kan et al. [12].

B. Subject Apps

To ensure that our results bear real-world significance, we have experimented with two real-life open-source Android products *K*-9 *Mail* [10] and *MyTracks* [15]. A summary of the subjects is shown in Table 1.

K-9 Mail [10] is a popular email client with more than 70,000+ lines of code and 40 test cases in its test suite (version 4.508). As shown in Google Play Store, the application has at least five million installs as at January 27, 2014.

MyTracks [15] is another popular app that tracks user activity using GPS sensors. It records movement statistics such as path, speed, and distance. The version used in our experiment (version 2.0.5) consists of 35,000+ lines of code and 347 test cases. These projects have also been used in the experiment in Li et al. [13] to evaluate energy-aware testing strategies. As shown in Google Play Store, the application has at least ten million installs as at January 30, 2014.

Table 1. Descriptive Statistics of the Subjects

Subject	Real-life version	SLOC	# of test cases	# of installs
K-9 *Mail* [10]	4.508	>70,000	40	5 million
MyTracks [15]	2.0.5	> 35,000	347	10 million

C. Preparation

Both subject apps utilize the standard Android test API which is based on JUnit. The test suites can be executed using the activity manager ("am") command via the Android Debug Bridge (ADB). For the experiments conducted in this paper, the subject apps have been preinstalled on the device under test. Our prototype implementation of CrowdAdaptor is split between an external Java app (external controller) and a preinstalled power management app (controller app). The external controller installs and controls the test suite executions, whereas the controller app adjusts the processor frequency range according to the external app, and records the energy consumption attributed to the test execution OS process.

To ensure compatibility across heterogeneous Android devices, the controller app has been adapted from two open-source Android projects *PowerTutor* [26] and *No-frills CPU Control* [16]. *PowerTutor* determines process-level energy consumption by periodically polling the processor utilization attributed to a process, and then translates it into energy values based on predefined power models represented by (1). *No-frills CPU Control* adjusts the frequency parameters passed to the CPU frequency policy governor as previously explained. The source code related to the required features has been identified and adapted to build the controller app.

D. Experimental Environment and Procedure

The sample device that we use for experimentation is a commercially available Sony Xperia SP smartphone equipped with a 1.7 GHz Qualcomm® (Snapdragon™ S4) MSM8980T Dual Core CPU running on Android 4.1.2. This family of smartphones has also been used in other energy-related studies [19]. We assume that a virtual machine of the same configuration is running on the device. The processor supports 14 operating frequencies from f_1 = 384 MHz to f_{14} = 1728 MHz. The firmware of the device has been updated to allow operations that require *root* access, namely, controlling the *ondemand* policy governor. *ondemand* is initially configured to use the entire interval of frequencies (that is, it can select any processor frequency between f_1 and f_{14}, inclusive). To avoid connectivity issues not directly related to the experimentation, we have opted to issue testing commands using ADB over a direct USB connection to a desktop computer. We note that remote execution is also supported by ADB over TCP/IP.

In order to evaluate the effects of different power models represented by (1), we have configured three separate executables of the controller app that compute linear, quadratic, and cubic relations of β_f with respect to *f*. Specifically, we have set $\beta_f = (f / f_1)^m$, where $m = \{1, 2, 3\}$, and set that $\beta_c = 0$ to focus our attention on the energy effects related to frequency selections as opposed to the power difference when a processor goes into a sleep state [26]. We note that these values may not represent the actual power model of a physical device. As a result, we will only report on relative energy consumptions rather than absolute values. As we have stated earlier in this paper, the actual values are likely to be somewhere among the three curves. For research purpose, since our target is to examine critically the energy saving potential of our proposed framework, we mainly look at the conservative side, which is the linear power model.

For each subject app, we have programmed the external controller to execute the test suite under the configuration for each power model and all the valid frequency subintervals. This allows us to compare the consumption values of any frequency range in the search space, and to simulate the *STDS*, *BTDS*, and *IRS* strategies.

In the case of K-9 Mail, each test suite execution completes within one second with negligible power dissipation. In order to reduce measurement errors and to obtain more accurate results, we have the external controller to construct a performance dataset *PD* for each frequency subinterval (line 13 of Algorithm 1) with 25 executions of the test suite (that is, *T′* has been reset 25 times in each configuration). In short, we have executed 387 (which is the sum of 40 and 347 for the two apps) test cases for each of 105 possible frequency subintervals, resulting in 141,435 sets of performance data.

After completing all the executions for a frequency interval, such statistics including execution duration, operating processor frequency, CPU utilization, and energy consumption estimations are written in log files stored locally on the device. The ADB interface is used to manually download the files for analysis. We have also built a data analysis tool to simulate *STDS*, *BTDS*, and *IRS* using the consumption values collected in the experiments. Specifically, we let each device complete the execution of a test case before a new test case is simulated. In summary, the globally minimal energy consumptions achieved in our experiments with K-9 Mail are 50.5%, 32.1%, and 17.4% of the energy consumed by the default configuration for linear, quadratic, and cubic power models, respectively. In the case of MyTracks, the global minima are 55.3%, 31.2%, and 6.2% of the energy consumed by the default configuration for each of the respective models. The results of each algorithm with respect to these global minima are evaluated in the next section.

E. Threats to Validity

Due to limited resources and the combinatorial effect of power models, frequency configurations, and subject apps, we are unable to expand the scale of the experiment, for instance, to support remote execution of more apps on multiple mobile devices with diverse VM types. On the other hand, we try to compensate by (a) ensuring that the prototype can be extended to support remote execution, (b) experimenting with Android, an open-source mobile platform that commands the market share of smartphones, and the same set of real-life subject apps used in a similar study [13], and (c) simulating heterogeneity in energy consumption by various power models previously used by other researchers. We have also spent our best effort to avoid program faults in our prototype implementation. Where possible, publicly tested third-party source code is adapted to implement the required features in the controller. The subjects used in the case study may not be completely developed by end-user programmers. A study of more relevant subjects should be conducted to generalize the result further. Our framework allows concurrent executions of test cases on different devices, and individual devices may query the configurations for their native execution. In the case study, we have not evaluated these aspects. We only measure the savings from the processor aspect while keeping other factors fixed or not monitoring them. Savings on non-virtualized components require further experimentation. We used *PowerTutor* and *No-frills CPU Control* to measure the energy and performance data. However, both of them only provide approximate data. Hence, our analysis results are affected by the accuracy offered by these two tools.

V. Data Analyses

Fig. 2 shows the energy consumptions of the three strategies expressed as a percentage of the default DVFS setting (that is, f_1–f_{14} for our case study). Since the algorithms have different terminating conditions, for fair and effective comparison, we plot the performance of the algorithms against the number of configurations compared. In the case of *IRS*, each data point reported in Fig. 2 corresponds to the average consumption after repeating the experiment 1000 times to average out the random factor at line 6 of Algorithm 4. The values on the *x*-axis (scaled to emphasize the trend of the first 20 configurations) correspond to the value of the parameter *max*. In Table 2, we show the performance of the algorithms and the minimum number of configurations compared to achieve the maximum savings. For instance, the cell for the *STDS* row of K-9 Mail shows that the maximum energy savings achieved by *STDS* is 3% after comparing 5 configurations. This corresponds to the point (5, 97) on the line of *STDS* in Fig. 2(a). The mean performance of each strategy (and each power model) is shown in the last sub-column of each major column (and in the last row, respectively).

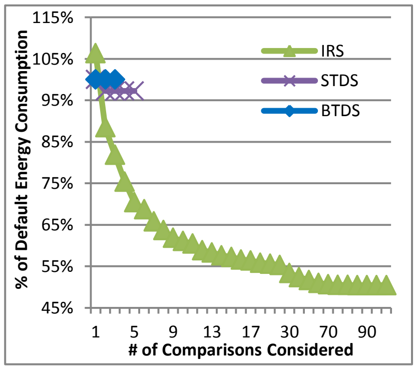
Since both *STDS* and *BTDS* start the search from the default frequency subinterval (that is, all the available frequencies), it is clear that their results will not exceed 100%.

We find from Table 2 that in four out of 18 cases (the cells that are not shaded in gray), the strategies do not discover significantly more energy-efficient configurations than the default *ondemand* policy governor. In particular, three of them belong to the *STDS* strategy. From subfigures (a), (c), and (e) of Fig. 2, we observe that in each of the four cases, the curves are quite straight and short. The poor performance of these cases may be explained by the application behavior that only a limited number of frequency usages can result in significant energy savings. However, these two algorithms require an improvement in energy savings for every other configuration tested. If the margins of comparisons are set too large, the algorithms may terminate early even though there are better configurations. This result indicates that when exploring the energy-efficient configuration space, an algorithm using a search-based strategy should consider its level of sensitivity.

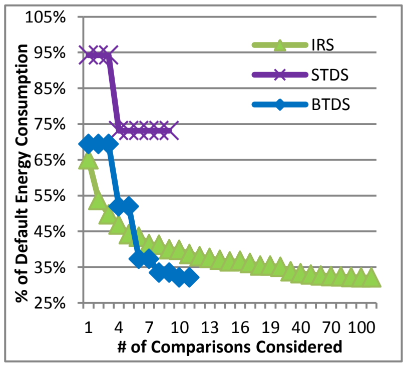
We now focus our discussion on the conservative power model (that is, the linear model) to assess the impacts of CrowdAdaptor on energy savings.

TABLE 2. THE MAXIMAL PERFORMANCE OF THE THREE STRATEGIES OF CrowdAdaptor IN THE CASE STUDY

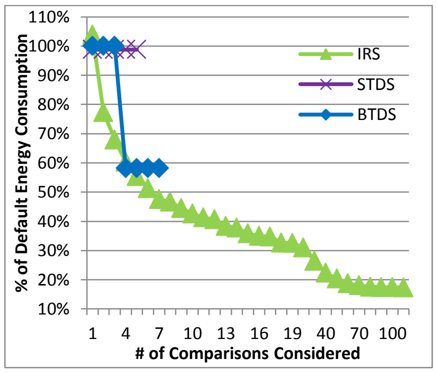
		Max. saving achieved in %				Min. # of configurations compared			
		Linear	Quadratic	Cubic	Mean	Linear	Quadratic	Cubic	Mean
K-9 Mail	*STDS*	3	27	1	10.3	5	9	5	6.3
	BTDS	0	68	42	36.7	3	11	7	7.0
	IRS	50	68	83	67.0	87	105	105	99.0
MyTracks	*STDS*	45	0	48	31.0	11	5	9	8.3
	BTDS	36	69	94	66.3	11	11	10	10.7
	IRS	45	69	94	69.0	105	105	105	105.0
	Mean	29.7	50.2	60.3	46.7	37.0	41.0	40.2	39.4



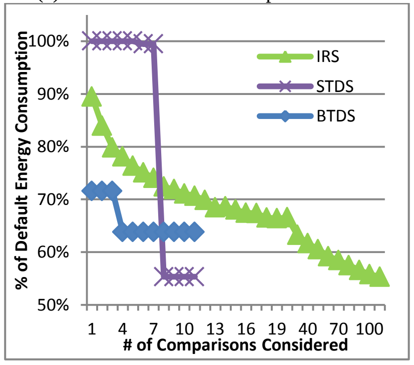
(a) K-9 Mail with linear power model



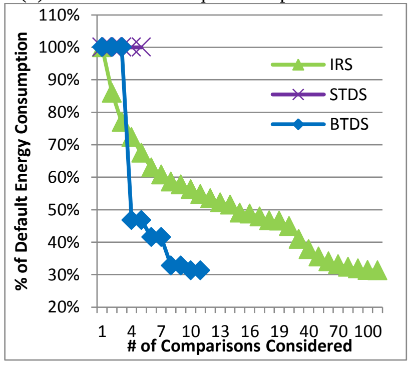
(b) K-9 Mail with quadratic power model



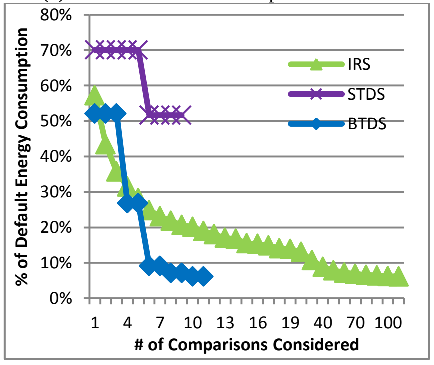
(c) K-9 Mail with cubic power model



(d) MyTracks with linear power model

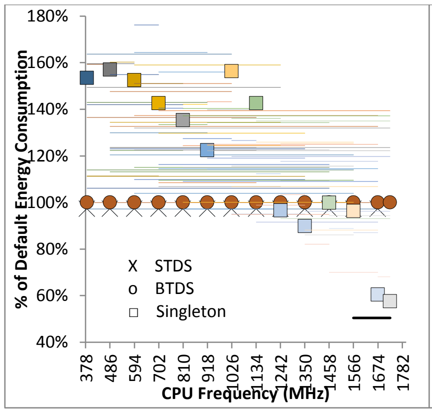


(e) MyTracks with quadratic model

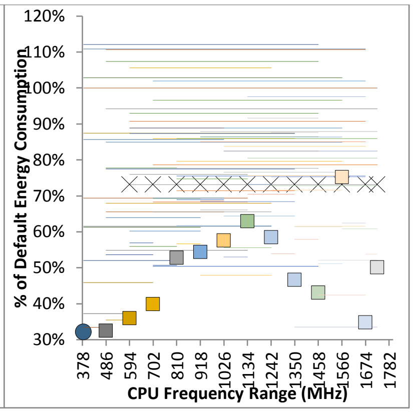


(f) MyTracks with cubic power model

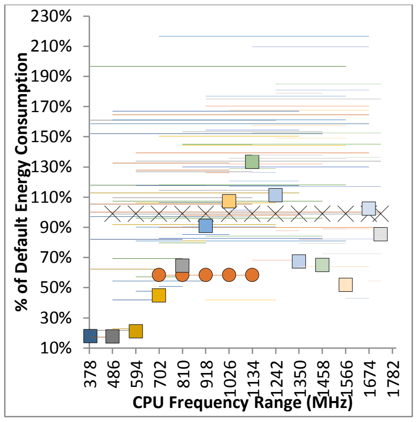
Fig. 2. Energy consumption and number of configurations compared.



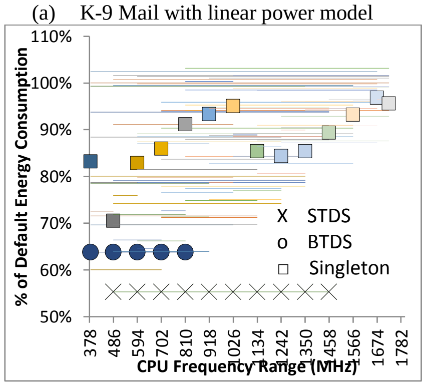
(a) K-9 Mail with linear power model



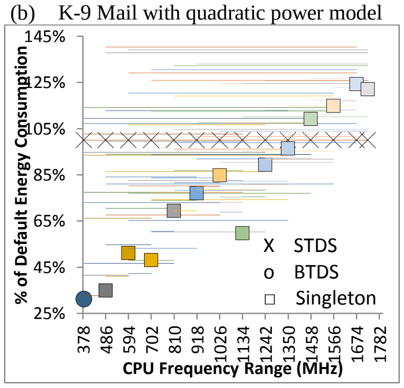
(b) K-9 Mail with quadratic power model



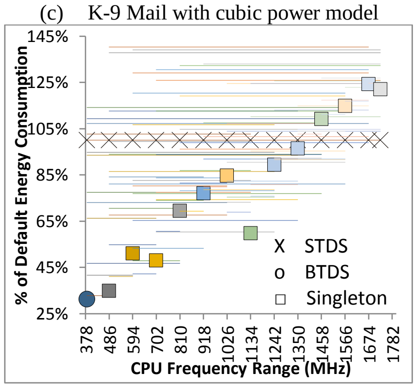
(c) K-9 Mail with cubic power model



(d) MyTracks with linear power model



(e) MyTracks with quadratic model



(f) MyTracks with cubic power model

Fig. 3 Energy consumption and number of executions compared.

As shown in Table 2, when *max* = 105 for *IRS*, CrowdAdaptor achieves a saving of 0–50% on K-9 Mail and 36–45% on MyTracks. The average saving is 29.7%, which is significant.

From subfigures (a) and (d) of Fig. 2, we also find that the numbers of configurations considered by *STDS* and *BTDS* before they terminate only differ by a small margin. In contrast, *IRS* can be instructed to search more configurations, but its effectiveness is realizable only if the participating devices can enumerate these configurations within reasonable time.

Next, we discuss the similarities and differences in terms of energy savings among the three power models on the three algorithms. We find from Table 2 that the savings on linear, quadratic, and cubic power models are 29.7, 50.2, and 60.5%, respectively, which are significant.

Across the rows, we find that *BTDS* achieves more savings than *STDS*. The minimum number of configuration trials needed to achieve the corresponding maximum savings also follows this relative order. For all the subfigures in Fig. 2 except subfigure (d), *BTDS* is the steepest, followed by *STDS* and then *IRS*. Therefore, *BTDS* is superior to *STDS* if fast convergence is required. In contrast, *IRS* needs to search an arbitrary number of configurations (or even enumerating all the available configurations) with slower convergence.

As a whole, the results show that each strategy and each power model exhibit significant impact on energy savings. However, we also observe that they exhibit significant differences in effectiveness. We thus further analyze the data to look into the variations in the dataset.

Fig. 3 shows the energy consumption for all 105 possible frequency subintervals on each subject using each power model. The horizontal bars span their represented frequency subinterval. Configurations that attribute to a fixed frequency (such as f_1–f_1) are represented by "□" markers and denoted by "Singleton" in the figure. The figure also shows the results of *STDS* and *BTDS* using "×" and "○" markers, respectively.

As seen in the figure, energy consumptions can vary greatly across frequency ranges. Even for singleton subintervals, the energy consumption may not have a definite correlation with the frequency, which means that the search for a good frequency subinterval, even with a limited subinterval length, is a nontrivial task. The minimum and maximum consumptions exhibit the smallest difference in subfigure (d), which range from 55 to 102% of the default configuration. *STDS* and *BTDS* perform better than half of the frequency subintervals, and achieve the global minima presented in Section IV.D in 4 out of 6 cases. For the purpose of baseline comparison, the global minima are identified by enumerating the energy consumptions of all the configurations.

In response to RQ1, the case study shows that even in the conservative (linear) power model, CrowdAdaptor is able to provide significant savings in processor energy consumption by an average of 29.7%.

In response to RQ2, the case study reveals that a higher order power model can result in more energy savings. The number of test executions does not seem to be significantly different. We do not observe significant differences in the trend of energy savings as more executions are used to locate better configurations, particularly between the curves for the quadratic and cubic power models.

VI. Related Work

The framework proposed in this paper and its application covers a number of research areas that are purely software (such as testing) or involve software/hardware integration. In this section, we review the results of some of the research work from each area.

A. Virtualization on Mobile Devices

Traditionally, the use of virtualization focus mainly on high end servers and data centers. Recent advancements in mobile computing resources have made system virtualization possible on tablets and smartphones [1][5]. Barr et al. [1] discuss the design and implementation of VMware's Mobile Virtualization Platform, and a use case to facilitate the Bring Your Own Device (BYOD) business policy [23]. The hosted (Type 2) mobile hypervisor is built on top ARMv7 processors for the virtualization of the Android OS. To cater for the security aspect of the business use case, an enterprise VM environment (maintained by corporate IT) is created on the mobile device with VPN tunnels formed in an isolated network namespace. Passwords and encryption are used to control access to the VM and local file storage. We believe that there will be increasing need for energy optimization of executions in VM as the technology for mobile virtualization matures.

B. Collection of Field Data from Deployed Software

Modern software systems are expected to deploy and operate in highly heterogeneous environments. Therefore, it can be very difficult to assess the quality and behavior in an in-house development environment. Orso [17] discusses the collection and analysis of field data from deployed software to tackle the problem. They propose a framework to augment in-house development tasks with field data. Using a remote agent and repository that reside in the deployment site, runtime field data are collected and transferred back to the developers for software maintenance and evolution. One of the usage scenarios is to leverage the large number of similar installations so that the burden of collecting field data can be shared by many groups of users, each responsible for monitoring a certain subsystem. The data collected can be used in software maintenance tasks and runtime improvements such as debugging, regression testing, performance tuning, and online failure recovery. The technique proposed in our paper shares a very similar vision as the one described in Orso [17]. In our paper, the field performance data collected from crowdsourcing are used by the CrowdAdaptor framework to gradually and automatically improve energy efficiency of all end users.

C. Energy Optimization in Testing

While optimization in energy consumption has been a popular topic of research, particularly in the field of mobile and pervasive computing, little effort has been spent to save energy in the context of software testing. One of the first efforts in this direction has recently been proposed by Li et al. [13]. In their work that addresses the problem of energy consumption in post-deployment testing, they propose a technique to eliminate test cases in the test suite such that the test coverage of the reduced test suite remains the same while energy consumption is minimized. Their work assumes that the number of test cases in the test suite can be reduced (and reduction is desirable) without compromising its effectiveness. Their approach is based on formulating test suite minimization as an integer linear programming (ILP) problem with the constraint that the coverage must not be reduced after the minimization. After collecting the coverage and energy consumption values, the problem is encoded and solved using an ILP solver. Their experimental results show that the technique can realize energy savings of up to 90%. Our paper tries to achieve a similar goal as Li et al., but our approach does not alter the test suite. On the other hand, our framework attempts to extract energy savings by switching power states, and does not make changes to the native computations.

Kan et al. [11] also address energy efficiency in testing and regression testing. The work investigates general and app-specific processor frequency assignment algorithms and their effectiveness in testing and regression testing. Two non-intrusive algorithms (that is, without changing the execution) are proposed for test suite execution. Since test suites are often executed many times in the process of software development, by setting different processor frequencies for the executions, an energy-efficient frequency can be determined per test case or per test suite. A software simulator-based experiment has been carried out to compare the effectiveness of using this frequency versus DVFS techniques proposed by other researchers. The paper reports that the single-frequency solution outperforms other techniques in energy savings, and is also efficient if carried forward to execute future versions of the same software in a regression testing scenario. The current paper is a generalization of this work and builds on top of the default *ondemand* Android CPU frequency policy governor. It considers not only single frequencies, but all the valid frequency ranges supported by the governor. Another improvement is that the empirical results are based on experimentation on a real mobile device rather than using a simulator.

D. Mobile Energy Optimization

Most of current research related to software energy consumption is in the area of mobile computing. Recent work by Hao et al. [8] focuses on estimating the energy consumption of mobile apps using program analysis. They propose a software-driven approach that measures energy usage of mobile apps at method, path, and source line granularities. The proposed technique assumes that there is an input workload to a mobile app for which an estimation of the energy consumption is required. The workload is then executed against an instrumented version of the app, which records path information and execution statistics, and computes the energy costs based on an instruction-level energy profile. Estimates are annotated in the source code for future visualization by the developer. It is reported from their experimentation results that the energy estimates are within 10% of the hardware-measured ground truth. Unlike this paper, the proposed methodology does not directly optimize the app or its executions. However, it can serve as a valuable tool allowing energy-aware developers to gain insights into the power consumption behavior, and make necessary energy optimizations prior to deployment.

Pathak et al. [20] presents another piece of work on energy profiling for mobile devices at process, thread, subroutine, and system call granularities. Similar to Hao et al. [8], the proposed technique requires instrumentation in addition to native routing and system-call tracing. The work proposes an accounting scheme for the energy consumed by wakelock-based components and concurrent access. In a case study, they found that 65–75% of energy consumed by free apps is accountable to third-party advertisements. Using the proposed profiler, they were able to identify several wakelock bugs and I/O energy bundles (I/O intensive periods), which add up to a significant amount of energy consumed. By visualizing the consumption in the context of bundles, the authors were able to rearrange and consolidate the energy consuming source code, and achieved 20–65% energy savings. The in-depth case study conducted in this work shows that energy optimization is of critical importance to mobile computing and requires substantial research effort.

VII. Conclusion and Future Work

In this paper, we have proposed the CrowdAdaptor framework. It is based on the provision of a number of factors: a large application installation base, the success of crowdsourcing systems and their contributors [6], and the availability of test cases during application development. It models a collection of energy efficiency settings as a configuration of a virtual machine that hosts an instance of the application. By leveraging the sheer size of the installation base, it spreads out the execution of the test cases to different configurations among all the devices willing to contribute. The paper also presents three strategies to explore the configuration space to locate configurations that are more energy efficient. The paper has presented a multi-subject case study to evaluate CrowdAdaptor. The results show that CrowdAdaptor is able to create additional power savings over the default CPU governor setting. To gain more insights into the effectiveness of CrowdAdaptor, we would like to expand the scale of the experimentation to include more complex apps and mobile devices with more diversified hardware specifications. In view of the recent trend of cloud-based mobile testing as a service, it will be interesting to apply the proposed framework to a cloud-based environment in order to study its feasibility and energy savings in a commercial environment. Another direction is to extend the usage scenario to other hardware components, and compare the effectiveness in energy savings with the existing power management techniques.

Acknowledgment

This research is supported in part by the Early Career Scheme and the General Research Fund of the Research Grants Council of Hong Kong (project nos. 111410, 123512, 717811, and 716612).

References

[1] K. Barr, P. Bungale, S. Deasy, V. Gyuris, P. Hung, C. Newell, H. Tuch, and B. Zoppis, "The VMware mobile virtualization platform: is that a hypervisor in your pocket?," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 4, 2010, pp. 124–135.

[2] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 2013.

[3] M. Burnett, C. Cook, and G. Rothermel, "End-user software engineering," *Communications of the ACM*, vol. 47, no. 9, 2004, pp. 53–58.

[4] G. Dhiman and T.S. Rosing, "Dynamic voltage frequency scaling for multi-tasking systems using online learning," *Proceedings of the 2007 International Symposium on Low Power Electronics and Design* (*ISLPED '07*), ACM, 2007, pp. 207–212.

[5] J.-H. Ding, C.-J. Lin, P.-H. Chang, C.-H. Tsang, W.-C. Hsu, and Y.-C. Chung, "ARMvisor: system virtualization for ARM," *Proceedings of the Linux Symposium*, 2012, pp. 93–107.

[6] A. Doan, R. Ramakrishnan, and A.Y. Halevy, "Crowdsourcing systems on the world-wide web," *Communications of the ACM*, vol. 54, no. 4, 2011, pp. 86–96.

[7] D. Ferreira, A.K. Dey, and V. Kostakos, "Understanding human-smartphone concerns: a study of battery life," *Proceedings of the 9th International Conference on Pervasive Computing* (*Pervasive '11*), Springer, 2011, pp. 19–33.

[8] S. Hao, D. Li, W.G.J. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," *Proceedings of the 2013 International Conference on Software Engineering* (*ICSE '13*), IEEE, 2013, pp. 92–101.

[9] H. Hoffmann, J. Eastep, M.D. Santambrogio, J.E. Miller, and A. Agarwal, "Application heartbeats for software performance and health," *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (*PPoPP '10*), ACM, 2010, pp. 347–348.

[10] *K-9 Mail*, GitHub Inc, 2014, https://github.com/k9mail/k-9/wiki/.

[11] E.Y.Y. Kan, "Energy efficiency in testing and regression testing: a comparison of DVFS techniques," The Symposium on Engineering Test Harness (TSETH '13), *Proceedings of the 13th International Conference on Quality Software* (*QSIC '13*), IEEE Computer Society, 2013, pp. 280–283.

[12] E.Y.Y. Kan, W.K. Chan, and T.H. Tse, "EClass: an execution classification approach to improving the energy-efficiency of software via machine learning," *Journal of Systems and Software*, vol. 85, no. 4, 2012, pp. 960–973.

[13] D. Li, C. Sahin, J. Clause, and W.G.J. Halfond, "Energy-directed test suite optimization," *Proceedings of the 2nd International Workshop on Green and Sustainable Software* (*GREENS '13*)), IEEE Computer Society, 2013, pp. 62–69.

[14] X. Liu, P. Shenoy, and M.D. Corner, "Chameleon: application-level power management," *IEEE Transactions on Mobile Computing*, vol. 7, no. 8, 2008, pp. 995–1010.

[15] *MyTracks for Android*, http://code.google.com/p/mytracks/.

[16] *No-Frills CPU Control*, Google, 2013, https://play.google.com/store/apps/details?id=it.sineo.android.noFrillsCPU.

[17] A. Orso, "Monitoring, analysis, and testing of deployed software," *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research* (*FoSER '10*), ACM, 2010, pp. 263–268.

[18] V. Pallipadi and A. Starikovskiy, "The ondemand governor: past, present, and future," *Proceedings of the Linux Symposium*, vol. 2, 2006, pp. 223–238.

[19] A. Paramanathan, M.V. Pedersen, D.E. Lucani, F.H.P. Fitzek, and M. Katz, "Lean and mean: network coding for commercial devices," *IEEE Wireless Communications*, vol. 20, no. 5, 2013, pp. 54–61.

[20] A. Pathak, Y.C. Hu, and M. Zhang, "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with Eprof," *Proceedings of the 7th ACM European Conference on Computer Systems* (*EuroSys '12*), ACM, 2012, pp. 29–42.

[21] *Power Profiles for Android*, Android Developers, https://source.android.com/devices/tech/power.html.

[22] N.B. Rizvandi, J. Taheri, and A.Y. Zomaya, "Some observations on optimal frequency selection in DVFS-based energy consumption minimization," *Journal of Parallel and Distributed Computing*, vol. 71, no. 8, 2011, pp. 1154–1164.

[23] B. Tokuyoshi, "The security implications of BYOD," *Network Security*, vol. 2013, no. 4, 2013, pp. 12–13.

[24] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation* (*OSDI '94*), USENIX Association, 1994, article no. 2.

[25] F. Xu, Y. Liu, Q. Li, and Y. Zhang, "V-edge: fast self-constructive power modeling of smartphones based on battery voltage dynamics," *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation* (*NSDI '13*), USENIX Association, 2013, pp. 43–55.

[26] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R.P. Dick, Z.M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," *Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (*CODES/ISSS '10*), ACM, 2010, pp. 105–114.