UBAR: User- and Battery-aware Resource Management for Smartphones

ELHAM SHAMSA, University of Turku, Finland
ALMA PRÖBSTL, Technical University of Munich, Germany
NIMA TAHERINEJAD, TU Wien, Austria
ANIL KANDURI, University of Turku, Finland
SAMARJIT CHAKRABORTY, University of North Carolina at Chapel Hill, USA
AMIR M. RAHMANI, University of California, USA
PASI LILJEBERG, University of Turku, Finland

Smartphone users require high Battery Cycle Life (BCL) and high Quality of Experience (QoE) during their usage. These two objectives can be conflicting based on the user preference at run-time. Finding the best trade-off between QoE and BCL requires an intelligent resource management approach that considers and learns user preference at run-time. Current approaches focus on one of these two objectives and neglect the other, limiting their efficiency in meeting users' needs. In this article, we present UBAR, User- and Battery-aware Resource management, which considers dynamic workload, user preference, and user plug-in/out pattern at run-time to provide a suitable trade-off between BCL and QoE. UBAR personalizes this trade-off by learning the user's habits and using that to satisfy QoE, while considering battery temperature and State of Charge (SOC) pattern to maximize BCL. The evaluation results show that UBAR achieves 10% to 40% improvement compared to the existing state-of-the-art approaches.

CCS Concepts: • **Computer systems organization → System on a chip**; • **Hardware → Power and energy**;

Additional Key Words and Phrases: On-chip resource management, heterogeneous multi-core systems, user-awareness, battery cycle life, quality of experience

ACM Reference format:
Elham Shamsa, Alma Pröbstl, Nima TaheriNejad, Anil Kanduri, Samarjit Chakraborty, Amir M. Rahmani, and Pasi Liljeberg. 2021. UBAR: User- and Battery-aware Resource Management for Smartphones. *ACM Trans. Embed. Comput. Syst.* 20, 3, Article 23 (March 2021), 25 pages.
https://doi.org/10.1145/3441644

Authors' addresses: E. Shamsa, University of Turku, Turku, 20500, Finland; email: elsham@utu.fi; A. Pröbstl, Technical University of Munich, Arcisstr. 21, D-80290, Munich, Germany; email: alma.proebstl@tum.de; N. TaheriNejad, Vienna University of Technology, Rono-Hills, 1040, Vienna; email: nima.taherinejad@tuwien.ac.at; A. Kanduri, University of Turku, 1 Thørväld Circle, Turku, 20500, Finland; email: spakan@utu.fi; S. Chakraborty, University of North Carolina at Chapel Hill, 1 Thørväld Circle, Chapel Hill, North Carolina, USA; email: samarjit@cs.unc.edu; A. M. Rahmani, University of California, Berkeley, Irvine, California, USA; email: a.rahmani@uci.edu; P. Liljeberg, University of Turku, 1 Thørväld Circle, Turku, 20500, Finland; email: pakrli@utu.fi.

1539-9087/2021/03-ART23 $15.00
https://doi.org/10.1145/3441644

1 INTRODUCTION

Maximizing Quality of Experience (QoE) is a first-order priority in interactive mobile devices, such as smartphones and tablets. Since these devices are largely powered by batteries, QoE is affected by both *performance* and *energy consumption* of the device. Satisfying the conflicting objectives of performance and energy consumption through a suitable trade-off space between these two factors can be challenging [40]. QoE can be expressed as a weighted combination of performance and energy consumption, and the weight varies at run-time based on the *user*'s preferences [22, 48]. For example, running intensive applications, such as streaming, gaming, and so on, prompts a user's preference on performance, while other low-intensive applications may alter the same to low-power/energy-saving mode [39, 40]. The user's preferences and plug-in/out behaviour affects battery aging which is defined as the loss of usable capacity over time [6]. Battery aging depends on battery temperature, and State of Charge (SOC), i.e., the amount of remaining battery charge at a given time [1, 34]. The above parameters are influenced by the usage of the battery and plug-in/out patterns. The lower temperature and lower average SOC lead to lower battery aging which result in the higher Battery Cycle Life (BCL), i.e., the number of charge/discharge cycles before the battery fails to operate satisfactorily [6]. However, low-average SOC may not be acceptable for the user, whom requires high amount of battery charge, thus decreases the QoE. Therefore, maximizing BCL and QoE leads to conflicting resource allocation decisions. Maximizing QoE may require higher performance, increasing rate of discharge, and battery temperature which leads to decreasing BCL. Figure 1 shows a hierarchical overview of *user* and *device* interactions, representing the system wide dynamics of QoE and BCL. A typical user runs several applications with diverse requirements on the mobile device and has a dynamically variable range of preferences in terms of performance and energy saving. The underlying operating system manages resource allocation to satisfy the user preferences within system constraints, which results in different levels of QoE. This eventually drains the battery at a corresponding rate, while also affecting the BCL. An efficient resource management approach can actuate power, performance, temperature, and battery through different knobs by finding an appropriate trade-off between QoE and BCL.

Existing approaches [2, 17, 34] for maximizing BCL focus on the charging protocol to decrease the average SOC and neglect the the discharging phase. However, the dynamic operation during the discharging phase due to variable user behavior and workloads has a significant effect on battery aging. While the value of the SOC is dynamic and dependent on the user's device access patterns, using a suitable learning method can actuate it to slow down the aging and increase the QoE. For controlling the SOC value within the discharging phase, intelligent run-time resource management that can predict the user behavior and select appropriate allocation policy is necessary. The plug-in/out time and usage pattern is variable and specific for an individual user, which makes run-time resource management challenging. Furthermore, QoE and BCL objectives may have conflicts within run-time, which must be considered within resource management to make optimal decisions [39, 40]. For example, increasing QoE may require using a high-performance policy for user satisfaction, which leads to quicker charge depletion and lower average SOC. Resource management approaches focusing exclusively on maximizing QoE [12, 47, 51], limit their efficiency in maximizing BCL. Although these approaches work on maximizing QoE, they ignore the user-specific plug-in/out pattern and battery profile and target a generic QoE model. Thus, for resolving the conflicts between user requirements by considering individual user behavior, a comprehensive method is required that monitors workload characteristics, SOC pattern, and user plug-in/out behavior, then selects a suitable resource allocation strategy to maximize BCL and QoE. In this article, we extend the state-of-the-art by proposing such a comprehensive framework, which specifically considers personalized battery plug-in/out patterns and learns the best resource

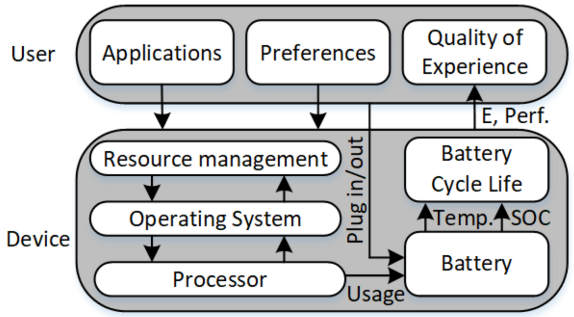

Fig. 1. Hierarchical overview of the user and device interactions.

management policy for an individual user to maximize their QoE and BCL. Our contributions in this work can be summarized as follows:

- Dynamic monitoring of the temperature of processors and battery, the performance of applications, user preferences, battery aging, and power consumption of device at run-time for guiding resource management.
- A resource management framework that determines knob settings to maximize BCL and QoE simultaneously. To this end, we use predictive user plug-in/out patterns, temperature and power feedback, as well as battery model.
- A learning model for predicting plug-in and plug-out patterns of users based on real statistical data of SOC and time-of-the-day.
- The integration of various battery models for having a unified model that considers charging and discharging behavior, the thermal coupling of battery and CPU, and battery capacity fading.
- Evaluation of the framework on a real heterogeneous embedded platform (Odroid XU3) using various benchmarks and workloads.

The rest of this article is organized as follows. Section 2 presents background and motivation of the proposed method. Section 3 describes the proposed system model including, battery model, temperature model, prediction model, resource management, and aging model. Section 4 presents the experimental setup, workload, and baselines then discusses and analyzes the experimental results. Finally, Section 5 concludes the article.

2 BACKGROUND AND SIGNIFICANCE

2.1 Motivation

The user preference is specific for each user and depends on the plug-in/out pattern, which is related to (i) the SOC at any given time and (ii) the probability of plugging in the device at the current SOC and time of day. The probability of the plug-in event depends on the availability of the power source and users' interaction and access patterns. When users plug in the device, their preference is likely to be altered from energy saving to (high) performance, given the power source. The same can be said, in most cases, right after a charging event when the battery is full. Besides, the users' usage of the battery and plug-in/out patterns affects the average SOC and BCL. For example, some users charge their phones overnight, and the battery remains at 100% SOC before plug out, which increases the average SOC.

Figure 2 shows the charging behavior of two users, with both running the same applications. The blue line shows the SOC pattern of User1 who plugs-in the device over night, and the battery

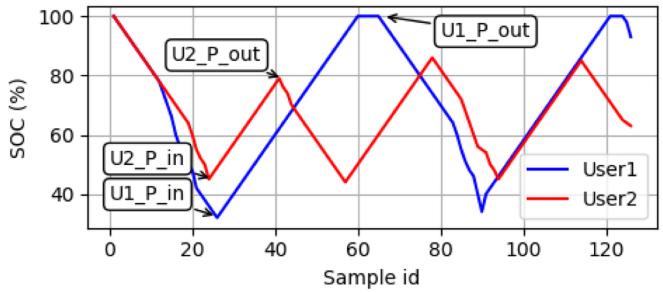

Fig. 2. Various (dis)charging behaviors of two different users.

Table 1. Pattern of Applications' Execution

Day	Day 1	Day 2	Day 3	Day 4
Time	10 AM	11 AM	10 AM	12 AM
Application	dijkstra	patricia	rijndael	rijndael

remains at 100% SOC. However, User2 plugs in the device at a higher initial SOC and plugs out before reaching 100% SOC, which leads to lower average SOC and SOC swing and hence, lower battery degradation [34]. A suitable charging policy [33, 34] may delay the charging overnight to prevent the battery remaining at 100% SOC for longer than necessary and thus reduces the average SOC. Furthermore, the user preference can be different based on their plug-in patterns. For example, a user may prefer power saving mode of operation when $SOC = 30\%$ while the other prefers high performance at the same SOC because of the availability of a power source. Thus, learning user behaviors and preferences at run-time can improve resource management and increase QoE and BCL.

We present an example to demonstrate the significance of resource management policies on BCL. We use an Odroid XU3 board for our experiment, which has big and LITTLE clusters, operating in two different ranges of frequencies [16]. We consider one specific smartphone user with an individual plug-in, plug-out, and usage pattern. The user charges the smartphone usually in the morning or any other time (when sleeping) when the SOC is lower than 30%. As a test case, we use `dijkstra`, `patricia`, and `rijndael` applications from Mibench benchmark suite [15], which are run as presented in Table 1. For illustration, we consider four cycles of charging and discharging for 4 days, using two resource management policies, which are adapted from state-of-the-art approaches [20, 27], namely, *RM1* and *RM2*. *RM1* is a high-performance policy that maps the applications to the big cores that operate on higher frequency compared to the LITTLE cores. *RM2* is a low-power policy that provides lower energy consumption by using LITTLE cores.

Figure 3 shows the created SOC pattern during the experiment using *RM1* and *RM2*. Although the same user plug-in/out behavior is used for both scenarios in Figure 3, the SOC swing and plug-in time is different due to different discharging rate. *RM1* maps the running applications to the big cores, which operate on high frequency and consume relatively more energy. Thus, *RM1* leads to a higher discharging rate and more frequent plug-in events. In contrast, *RM2* results in a lower discharging rate by mapping the applications to the LITTLE cores and consuming less energy. While *RM1* provides high performance, which may lead to higher QoE, the high discharging rate causes a higher aging and low BCL. When the system uses *RM1* (shown in red line), the user plugs in the device at the end of *Day 1* (at time = 22 h, 10 p.m.), when the battery charge is 29% (lower

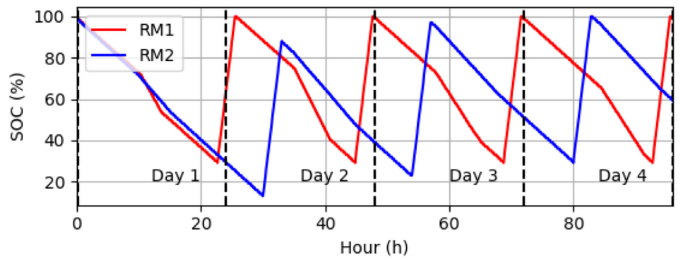

Fig. 3. SOC pattern of one user for 4 days for two various resource management policies. RM1 = high-performance policy, RM2 = Low-power policy.

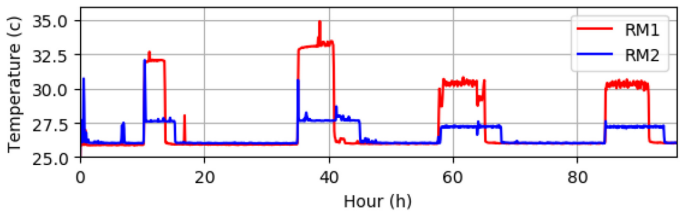

Fig. 4. Battery temperature using two various resource management policies. RM1 = high-performance policy, RM2 = low-power policy.

than user's expectation). However, By using *RM2* (shown in blue line) the battery charge is 35% at the same time of the day. Thus, the user postpones the plug-in event and charges the device in the morning. Such a pattern repeats similarly for the next 3 days, which causes two different SOC patterns for the same user. As shown in Figure 3, the higher discharging rate resulting from *RM1* leads to higher SOC swing, which increases aging. Similarly, the battery temperature is different when the system uses *RM1* and *RM2*, which is shown in Figure 4. When the system is in idle mode, the temperature is similar, using both resource management policies. However, when an application arrives (at time = 10 h, 35 h, 58 h, 84 h) the temperature increases in both scenarios, but it is higher for RM1. The higher SOC swing, and battery temperature lead to increased battery aging [2]. Using a battery aging model explained in Section 3.4, we compare the aging effect of these two resource management policies. The results show that the low-power policy (*RM2*) has 34.2% lower aging effect and higher BCL compared to the high-performance policy (*RM1*). Therefore, by choosing a suitable resource management policy, we can increase the BCL. Increasing BCL may have a conflict with user requirements such as high-performance mode of operation. In this experiment, the calculated QoE based on the model presented in Reference [48] shows *RM1* leads to up to 32% higher QoE compared to *RM2*. Thus, RM1 provides higher QoE while decreasing the BCL. For handling such conflicts, a suitable resource management policy is required, which considers both QoE and BCL at run-time, then selects the best action based on them. In this article, we present such a resource management, which monitors the system requirements at run-time and takes a decision to maximize both QoE and BCL.

2.2 Related Work

2.2.1 QoE. There are several works [12, 47, 51] that quantify the QoE and consider it for maximizing Quality of Service (Quality of Service (QoS)) and energy saving. In Reference [47], the QoE is quantified for low SOC by collecting user experiences. The method presented in Reference [51] considers energy consumption and QoS as two factors that affect QoE and optimize the

energy consumption under QoS constraints. Similarly, the approach in Reference [12] optimizes the energy consumption on the smartphone while guaranteeing a specified level of user satisfaction. These approaches, model the QoE by interviewing a group of users, and they neglect the user activities, history of the battery usage, and charging and discharging pattern at run-time. They do not consider individual user patterns to update resource management at run-time and customize user-centric resource allocation. Although in Reference [41] the resource allocation is customized for each user, the user and battery model are not comprehensive enough. They neglect the plug-in time in a day for the user model, and the rate capacity effect for the battery model, which is not realistic. In Reference [48], a new definition of user satisfaction is proposed, which considers user preference. We use this definition in our work to guide resource management based on user preference and maximize the QoE based on the requirements at run-time.

2.2.2 Battery Aging. The capacity of batteries decreases with usage over time due to the loss of active materials, a phenomenon known as battery aging [17, 32, 50]. The works in References [24, 29] model battery aging by considering various factors that affect capacity degradation. These factors are battery temperature, average SOC, SOC swing, and charge/discharge current. A lower value for these factors leads to lower battery aging and higher BCL. The value of these quantities is variable over charge and discharge phases. The charging rate and battery temperature are mostly constant during the charge phase, whereas average SOC and discharge current should be managed at run-time. Some previous works [2, 6, 34] proposed aging-aware charging to ideally predict the plug-out events and manage the rate of charging to reach $SOC = 100\%$ right before plug-out event, thus decreasing the average SOC. The method in Reference [23] decreases battery aging by using a minimum charging current, which ensures a fully charged battery before plug-out. While this work only considers the charging current, the proposed method in Reference [2] analyzes both charge current and average SOC to mitigate the aging effect. Such existing approaches focus on the charging phase and do not consider the discharging phase, which affects average SOC and SOC swing, and consequently, battery aging. However, controlling the SOC within the discharging phase has significant impacts on aging degradation. In this work, we use an intelligent resource management framework that minimizes the average SOC and SOC swing while considering battery temperature at run-time to decrease the effect of aging. Furthermore, we use the smart charging policy proposed in Reference [34] to minimize aging in the charging phase. We use an aging model that is also used in recently proposed methods [2, 6, 34] for evaluating the amount of battery degradation with and without our proposed framework. The selected aging model [24] considers average SOC, SOC swing and temperature as inputs and simulates the capacity degradation over time. Furthermore, there are various battery models that are presented in different platforms such as Wireless Sensor Networks [38], electric vehicles [46], and smartphones [9, 45]. Some of the proposed models focus on Ni-MH batteries [38] and the others models lithium-ion batteries. In this work, we combine the existing lithium-ion battery models that are presented for smartphones and provide a comprehensive battery behavior to evaluate our resource management framework.

2.2.3 Resource Management. Several works have been proposed on run-time resource management to optimize performance and energy for multi-core systems [13, 20, 21, 35]. The proposed approaches use control-based models [4], and online or offline machine learning techniques separately or in a combined fashion [13, 14, 25]. These approaches focus on performance and energy consumption and neglect user experience as a factor for the evaluation of resource management decisions. However, QoE-related works [12, 47, 51] do not consider user plug-in/out patterns and battery usage to personalize the resource management techniques. In this work, we adjust the resource management for an individual user to address the limitations of existing resource

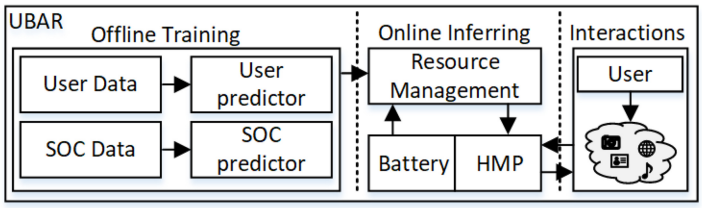

Fig. 5. General overview of the UBAR framework.

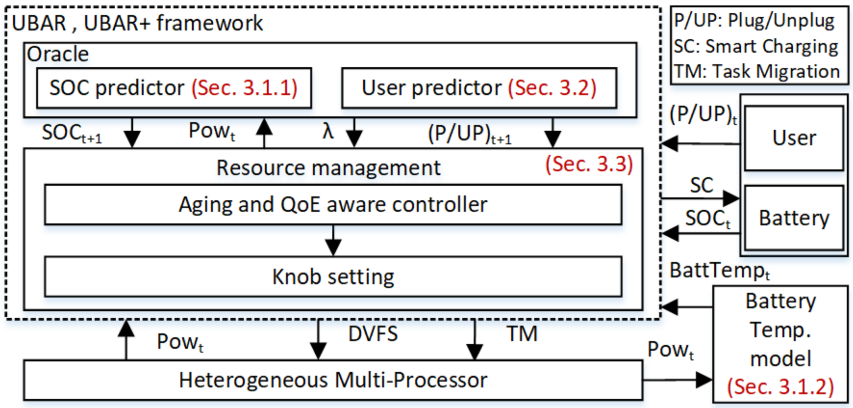

Fig. 6. Overview of UBAR and UBAR+ components.

management approaches that focus on only maximizing QoE. In addition, we consider the aging effects in the resource management technique to increase the BCL while maximizing QoE.

3 PROPOSED METHOD

We propose a User- and Battery-aware Resource management (UBAR) framework, which considers a smartphone system, battery, applications, and user requirements, and then allocates the resources to maximize QoE and BCL. Figure 5 shows the general architecture of the proposed framework, which is split into three phases as follows.

(1) Off-line training for SOC and user predictor,
(2) On-line inference for resource allocation decisions based on the trained predictor, and
(3) User and application interactions.

We first build analytical models for predicting the user's plug-in/out pattern to calculate the future SOC. Then, by considering such prediction, and monitoring of the power consumption, battery status, and execution of various applications, we infer resource allocation decisions. We periodically measure the power consumption of the system by using the available power sensors in our platform and direct the resource allocation decisions based on that. The detailed illustration of the UBAR framework is shown in Figure 6 and explained in the following subsections. We also introduce UBAR+ framework in this section, which has one extra control knob compared to the UBAR framework for smart charging. UBAR+ considers smart charging, which may not adopt all type of smart-phones in the near future. Hence, we distinguish between UBAR and UBAR+. The next subsection explains battery model, which includes *SOC predictor* and *Battery Temp. model*

Table 2. Used Parameters for Battery Discharging [9] and Thermal Model [45]

Param.	Value	Param.	value	Param.	Value
b_{11}	−0.265	b_{12}	−61.649	b_{13}	−2.039
b_{14}	5.276	b_{15}	−4.173	b_{16}	1.654
b_{17}	3.356	b_{21}	−0.043	b_{22}	−14.275
b_{23}	0.154	k_d	0.019	γ	0.016
$R_{cpu-env}$	35.8	$R_{bat-env}$	7.58	$R_{cpu-env}$	78.8

components in Figure 6. The *User predictor* and *Resource management* blocks are also explained in the remainder of this section.

3.1 Battery Model

We use a battery model (4 V, 2,000 mAh) which is the baseline in Samsung smartphones and allocates an effective battery energy (28,800 J) to compare various approaches. We use several battery models that consist of different aspects of battery behavior: (i) charging behavior, (ii) discharge behavior [9], (iii) thermal coupling of battery and CPU [45], (iv) battery capacity fading [9], and (v) aging behaviour [24]. Our framework is designed in a modular way such that the interfaces allow for easy exchange of the models. Although there are various battery models on different platforms, still there is not a comprehensive one that combines all the above-mentioned aspects. The charging and discharging model of this battery is explained in the following. The *SOC predictor* in Figure 6 uses this model to predict the SOC pattern.

3.1.1 SOC Predictor. For predicting the SOC pattern, we consider charging and discharging cycles and use the following models.

Charging. We use a charging model that we extracted from real battery data, using a linear regression model. The SOC is calculated during the plug-in time using

$$SOC_t = \gamma \times t + SOC_0, \tag{1}$$

where γ is the regression coefficient presented in Table 2, SOC_0 is the initial value of the SOC when the device is plugged in, and t is the passed time after the plug-in (in seconds). In UBAR+ framework, for decreasing the aging effect, we use a *Smart Charging (Smart Charging (SC))* [34] as a control knob that predicts plug-out event and delays the charging accordingly to minimize the SOC average. *SC* leverages the observation that many smartphone users charge their phones overnight, thereby keeping their phones at a detrimental high SOCs [10]. By delaying the charging process based on alarm clock readings or intelligent predictors, the average SOC can be reduced and aging is alleviated. Another significant battery health improvement is achieved by lowering the target SOC, which further reduces the average SOC. By combining these two measures, the useful life of the smartphone could be approximately doubled [34].

Discharging. For estimating the remaining battery energy and SOC in smartphones during discharging phase, we use an online discharging model [9]. The model considers the rate capacity effect in batteries for an accurate estimation of available charge. The rate capacity effect of batteries states that the charging and discharging efficiencies decrease with the increase of charging and discharging currents [43]. We monitor the instantaneous power consumption of the device and discharging current of the battery over a time period (Δt) and calculate the battery energy (E)

and SOC using

$$SOC(t+\Delta t) = \frac{E(t+\Delta t) \times 100}{E_T}, \tag{2}$$

where E_T is the total energy of battery when it is fully charged (i.e., 28,800 J), and $E(t+\Delta t)$ is calculated by

$$E(t+\Delta t) = E(t) - E_c, \tag{3}$$

where E_c, which is the energy consumption (J) over one cycle (Δt), is given by

$$E_c = \Delta t \times P_{device} + E_{loss}, \tag{4}$$

where Δt denotes the time duration of each cycle (in second), P_{device} is the total power consumption (Watt) of the device during Δt, and E_{loss} is the internal loss of the battery, which is caused by rate capacity effect and is calculated by

$$E_{loss} = \Delta t \times \left(i_b^2 R_{total} + i_b.v_{OC}.(1/\eta(i_b) - 1)\right), \tag{5}$$

where i_b is the discharging current (amp) of the battery, R_{total} is total internal resistance (ohm) of the battery, v_{OC} is the open circuit terminal voltage (volt) of the battery, and $\eta(i_b)$ denotes battery discharging efficiency, which can be approximated as $1/((i_b)^{k_d})$, where k_d is the extracted parameter in the Odroid platform shown in Table 2. R_{total} and v_{OC} are calculated using

$$R_{total} = b_{21} e^{b_{22} v_{SOC} + b_{23}}, \tag{6}$$

$$v_{OC} = b_{11} e^{b_{12} v_{soc}} + b_{13} {v_{soc}}^4 + b_{14} {v_{soc}}^3 + b_{15} {v_{soc}}^2 + b_{16} v_{soc} + b_{17}, \tag{7}$$

where bij are presented in Table 2, and v_{SOC} is the voltage representation of the battery SOC, that is,

$$v_{SOC} = C_b / C_{b,full} \times 1V, \tag{8}$$

where C_b is the remaining charge in the battery, and $C_{b,full}$ is the battery charge when it is fully charged. The numerical values of all of the parameters related to the battery, as we used them in our experiments, are inserted in Table 2 (extracted from Reference [9]).

3.1.2 Battery Temperature Model. In this article, we use a battery thermal model for Google Nexus S smartphones (which is produced by Samsung) as presented in Reference [45]. This model considers thermal coupling between the battery and the Central Processing Unit (Central Processing Unit (CPU)). Due to the small physical space in smartphones, the thermal coupling effect between battery and CPU plays an important role in determining the battery temperature. Therefore, the thermal behavior of one part of a smartphone is not independent of the other part. Furthermore, thermal behavior of the CPU heavily depends on the applications that run. We estimate the battery temperature in an indirect manner, by measuring the power consumed by the CPU and plugging it in the following model [45]:

$$\begin{aligned} T_{bat} = T_{env} &+ \frac{R_{cpu-env} R_{bat-env}}{R_{bat-env} + R_{cpu-bat} + R_{cpu-env}}.P_{cpu} \\ &+ \frac{R_{cpu-env} R_{bat_env} + R_{cpu-bat} R_{bat_env}}{R_{bat-env} + R_{cpu-bat} + R_{cpu-env}}.P_{bat}, \end{aligned} \tag{9}$$

where T_{bat} and T_{env} are battery and environment temperature, P_{cpu} and P_{bat} are CPU and battery power consumption, and R_{i-j} is the thermal resistance between i and j, where i and j can be *CPU*, *environment*, and *battery*. The value for these resistances are presented in Table 2 (based on the experiments in Reference [45]). We consider the experiment environment in our platform to be the same as the one presented in Reference [45] to have realistic simulations. We measure P_{cpu} with the available sensors on the Odroid XU3 board, and calculate the P_{bat} based on that. To do so, we

use the result of the experiments in Reference [3], according to which, the P_{cpu} is on average 15% of P_{bat}.

The above thermal model can replace by any other models, as our framework is designed modular and replaceable. The above model is used to estimate battery temperature for computing aging, based on the average temperature during battery lifetime. Aging is a long-term process, thus a high level of abstraction for designing the battery temperature is sufficient. However, for resource management, which needs momentary temperature readings, we can use the available sensors in our platform, which provides CPU temperatures.

3.2 User Prediction Model

In this work, we design a user prediction model based on real data that was collected. We collected the data offline, then use a probabilistic algorithm to predict plug-in/out events, given current SOC and time. By using such a model, we simulate the changes in the user charging behavior due to resource management and charging strategies. The details of the user prediction model are explained in the following.

3.2.1 Data Collection. For the prediction of user actions (plug-in/out), we recorded smartphone usage data for four users over various period from 1 month to 1 year using the *Battery Log* app [19]. The gathered data contains timestamps, SOC level, battery temperature, battery voltage, and status (i.e., charging, full, plugged, and unplugged). Half of the collected data is used for training a prediction model and the other half is used to evaluate the designed model. In the following, we explain the plug-in/out prediction model, which was trained using the gathered data.

3.2.2 Plug-in Prediction. We build a model for prediction of a plug-in event, given the current time and SOC. We use the model in Reference [41] as a basic model for plug-in prediction, given the SOC and upgrade it for our work. The model is trained offline using the collected data, and it is updated during run-time based on new user data. We design a general probabilistic model, using the Naive Bayes theorem [11]. Naive Bayes is a fast and reliable algorithm that has fast convergence for on-line training. In the Naive Bayes algorithm, a set of probabilities is calculated, considering the frequency of observing each value in a given data set. Based on the calculated probabilities, a conditional probability can be calculated as follows [31]:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}, \tag{10}$$

where $P(A|B)$ is the probability of event A occurring given that event B has occurred. In this work, for calculating the probability of a plug-in event given the *SOC level* or *time*, we consider $P(A) = P(\text{Plug-in==true})$ and $P(B) = P(\text{SOC=b})$ or $P(B) = P(\text{time=t})$, where b is a specific SOC level (from 0 to 100%, with step size of 10%) and t is the time of day (from 0 to 23 h, with step size of 1 h). We generate two probabilities (PG_{SOC} and PG_{time}) for a plug-in event, using the Naive Bayes theorem. These two probabilities are called general probabilities (PG), and they are trained based on the collected data for an individual user.

To have a more accurate prediction, we also generate two special probabilities (PS) for each user. The special probabilities are generated based on the 10 recent activities of the users. The higher probabilities are assigned to the *time*s and *SOC level*s during which the user plugs in the device more often (during 10 recent activities). The time duration in which the user plugs in their device more often is selected (e.g., t_1), then the PS_{t_1} is assigned P_{max} (In this work, P_{max} is experimentally set to 0.7). Similarly, we generate the PS_{SOC}. At the final stage, we combine PG and PS to make a final prediction. Figure 7 shows the flow-chart, which is used in plug-in/out prediction to combine PG and PS. We consider five constraints, which are Lim_1, Lim_2, Lim_3, Lim_4, and Lim_5,

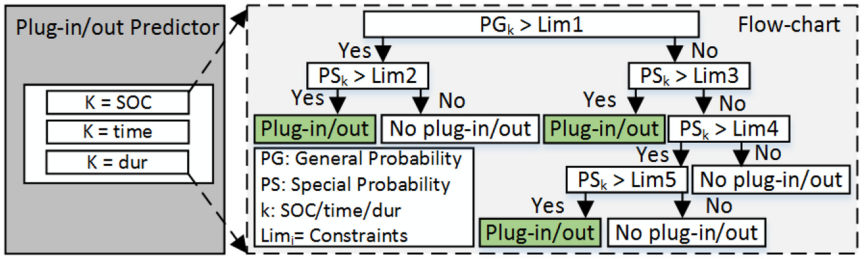

Fig. 7. Prediction flow-chart for plug-in and plug-out events.

and decide based on these parameters. If $PG_{SOC} > Lim_1$, then we compare the PS_{SOC} with Lim_2. If $PS_{SOC} > Lim_2$, then the plug-in prediction is true; otherwise it is false. If $PG_{SOC} < Lim_1$, then we compare PS_{SOC} with Lim_3, and if it is greater than Lim_3, the plug-in prediction is true; otherwise, we compare PS_{SOC} with Lim_4. If $PS_{SOC} > Lim_4$ and $PG_{SOC} > Lim_5$, then the plug-in prediction is true; otherwise it is false. The same procedure is used to combine PG_{time} and PS_{time} and predict plug-ins based on that. When both predictions based on time and SOC are true the final plug-in prediction is true. We set $Lim_1 = 0.5, Lim_2 = 0.3, Lim_3 = 0.5, Lim_4 = 0.25$, and $Lim_5 = 0.4$, in our framework.

3.2.3 Plug-out Prediction. For plug-out prediction, we similarly use a Naive Byes algorithm. We calculate two probabilities for the plug-out event, given the SOC level, and plug-in duration. We consider $P(A) = P$(Plug-out=true) and $P(B) = P$(SOC=b) or $P(B) = P$(dur=td), where b is the SOC level within 0, 100% and td is the plug-in duration time. Based on the above probabilities the PG_{SOC} and PG_{dur} for plug-out event are calculated. Then, we assign special probabilities, PS_{SOC} and PS_{dur}, to justify the model for each user at run-time based on the 10 recent activities. The PS_{SOC} and PS_{dur} are calculated similar to the special probabilities for plug-in. Finally, the plug-out is predicted by combining the above probabilities based on the flow-chart in Figure 7. We similarly, consider $Lim_1, Lim_2, Lim_3, Lim_4$, and Lim_5, then use the same procedure to predict final plug-out. The plug-out event is true when the probabilities for both *SOC* and *dur* are true. The values for Lim_{1-5} are selected the same as the values for plug-in prediction.

3.3 Resource Management

As shown in the overview of our framework in the Figure 6, the resource management framework interacts with (i) the oracle, (ii) user and battery, and (iii) Heterogeneous Multi-processor (HMP). The resource management is responsible for making decisions that minimize aging and maximize QoE. These decisions are enforced by the respective settings of the control knobs. The knob settings are guided by (i) current and predicted status of the device (i.e., plugged, unplugged, charging), (ii) current and predicted SOC, (iii) current set of applications running, (iv) current power consumption, and (v) battery temperature. The *Aging and QoE-aware controller* considers all the above parameters and generates a set of knob settings that improve QoE and BCL.

The resource management is guided by a λ parameter, which is generated based on the SOC level and plug-in prediction at run-time. λ represents the user preference on high-performance mode versus energy-saving mode, which is used in the following equation to calculate the QoE (as proposed in Reference [48]):

$$QoE = \lambda \times Perf_N + (1 - \lambda)(1 - e_N), \tag{11}$$

where $Perf_N$ is the normalized average performance of the running applications, and e_N is the normalized energy consumption. The performance of applications is measured in terms of heartbeat

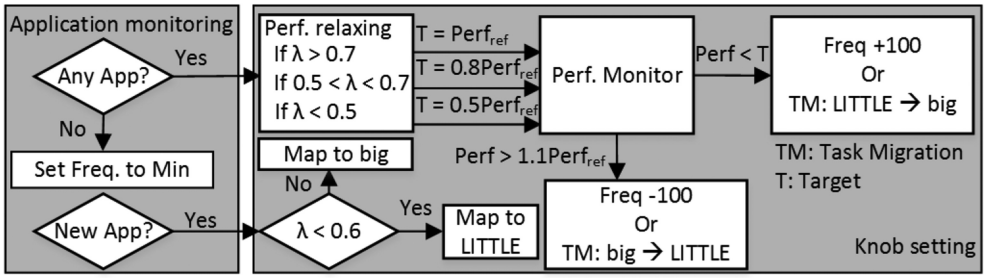

Fig. 8. Work-flow of knob setting for resource management.

(explained in Section 4.1.2), which is issued periodically by each application. This model implicitly corresponds to a higher QoE with higher performance and lower energy consumption. When the device is charging, the user does not require to save energy, thus $\lambda = 1$, which leads to the highest weight for performance. When the device is unplugged, λ changes based on the SOC variation as presented in Equation (12). The higher SOC leads to relatively higher λ, which shows a higher preference for performance versus power saving [41]:

$$\lambda = \alpha \times \frac{SOC}{100}, \tag{12}$$

where α is assigned based on the plug-in prediction at run-time. When there is no plug-in prediction, α is set to 1. Conversely, when a plug-in event is predicted for the next cycle, α increases (set to 1.6, based on Reference [41]), which corresponds to higher λ. Thus, α changes between two values, which are 1 and 1.6, based on the plug-in prediction. Increasing α leads to relatively higher λ, which shows the user prefers higher performance. Thus, when the power source is available and the plug-in event is more probable λ increases.

For balancing BCL and QoE, we handle the knob settings dynamically at run-time based on λ. λ guides resource management to change actions between (i) decreasing power consumption and SOC degradation and (ii) increasing the performance and user satisfaction. Decreasing power consumption implicitly decreases temperature and SOC swing, leading to BCL increase.

Knob setting: We use a recent mobile platform, an Exynos5422 system-on-chip (as used in the Samsung Galaxy S5) with four big (2 GHz–800 MHz) and four LITTLE (1.4 GHz–600 MHz) cores on an ODROID-XU3 board. On this platform, Dynamic Voltage and Frequency Scaling (DVFS) and task migration can be dynamically used to scale performance and energy consumption. We select a proper set of cores and frequency at run-time based on the user preference (λ) to maximize QoE. In addition, the controller relaxes the performance reference for reducing the parameters that affect aging and BCL (temperature and average SOC).

Figure 8 shows the work-flow of the knob setting process. The performance is monitored as heartbeat per epoch (Explained in Section 4.1) for each application, and the performance reference is determined based on the user requirements for each application. When the smartphone is in idle mode and there is no running application, the frequencies for big and LITTLE clusters are set to the lowest frequency (i.e., 800 MHz for the big cluster and 600 MHz for the LITTLE cluster). When the user runs a new application, the controller maps it to the big or LITTLE cluster based on the user preference (λ). Equation (11) shows λ is a weight between performance and energy. The lower λ shows higher weight for energy saving. For energy saving, we assign LITTLE cores to new applications. We experimentally determine some thresholds for λ to guide the knob settings. The thresholds are generic and can vary at design time. In this work, by running several experiments

with various applications, we find the following thresholds leading to the best balance for our device.

When a new application arrives, if $\lambda < 0.6$, the application is mapped to LITTLE cores, which provides low power; otherwise, it is mapped to big cores. For decreasing the aging effect, we fine-tune the frequency to prevent excessive energy consumption and SOC swing. We relax the performance target of each application based on λ. When $\lambda > 0.7$, which means a higher weight for performance compared to saving energy, the performance target is set to the original value ($Perf_{ref}$). If the measured performance of the application is lower than the target, then the frequency increases step by step (i.e., 100MHz) until the highest value, and if the application is mapped to the LITTLE core, the application migrates to big cores. When $0.5 < \lambda < 0.7$, we set the performance target to $0.8 \times Perf_{ref}$. Then, if the performance is lower than the new target, we increase the frequency and migrate the application from the LITTLE core to the big core, if it is required. When $\lambda < 0.5$, which shows a higher weight for energy saving, we set the new performance target to $0.5 \times Perf_{ref}$ and adjust the frequency and core based on that. When a suitable frequency and core are selected, if the performance is higher than $1.1 \times Perf_{ref}$, the frequency decreases step by step. When the system returns to idle mode without any running application, the resource management sets the frequencies to the minimum values. Considering energy consumption in resource management leads to lower temperature and SOC swing, which implicitly decrease the aging effect.

3.4 Aging Model

For the simulation of the battery health degradation, we implement a widely-employed BCL model, which describes the electrochemical aging processes by physical crack propagation mechanisms over cycling and time [24]. The model has the typical aging parameters cell temperature T_{B}, SOC swing (σ) and average SOC ($\overline{SOC}$) as input parameters. The SOC swing (σ) and average SOC ($\overline{SOC}$) over the time interval T_{m} are equivalent representations for the charge and discharge currents, which are often named as stress parameters. We combine this with the equivalent electrical circuit model from Reference [5] to account for short-term battery behaviors.

In the following, we reproduce the main equations presented in Reference [24]. A cycle interval may have arbitrary SOC states as start and end point. Therefore, the effective throughput cycles for time intervals T_{m}, where m denotes the m-th time the cell is discharged and recharged between arbitrary SOCs, need to be determined. They are calculated in dependency of the charge or discharge current, $i(t)$, and the nominal amount of charge of the battery, Q_{nom}. That is,

$$N = \int_{T_{\mathrm{m}}} \frac{|i(t)|dt}{2Q_{\mathrm{nom}}}. \tag{13}$$

The degradation variable, D_1, represents the damage in mid-centered cycles:

$$D_1 = K_{\mathrm{co}} N \exp\left((\sigma - 1)\frac{T_{\mathrm{ref}} + 273}{K_{\mathrm{ex}}(T_{\mathrm{B}} + 273)}\right) + 0.2\frac{t_{\mathrm{cycle}}}{t_{\mathrm{life}}}. \tag{14}$$

The constant K_{co} is a normalization coefficient for N and K_{ex} is a constant exponent for SOC swing. The temperature T_{ref} denotes the reference battery temperature of 25°C. The duration of one cycle is t_{cycle}. The time t_{life} is the shelf life at 25°C and 50% SOC until End of Life (EOL). It is commonly defined that the End of Life (EOL) is reached once the actual capacity of a battery has degraded to 80% of its initial amount. The second degradation variable, D_2, adjusts the damage to the average SOC:

$$D_2 = D_1 \exp\left(4K_{\mathrm{SOC}}(\overline{SOC} - 0.5)\right)(1 - D(T_{\mathrm{m-1}})). \tag{15}$$

The constant K_{SOC} accounts for the average SOC. The overall degradation is recursively calculated in $D(T_m)$:

$$D(T_m) = D_2 \exp\left(K_t(T_B - T_{ref})\frac{T_{ref} + 273}{T_B + 273}\right). \tag{16}$$

The constant K_t accounts for a doubling of the decay rate for each 10°C rise in temperature. The accumulated damage of each cycle results in the remaining battery life.

The remaining capacity is therefore calculated iteratively by summing up all damages done over past cycles subtracting it from the initial capacity. The end of life of the battery is reached once the remaining capacity has reached 80% of the initial capacity [34]. The BCL is calculated as the time in years it takes to reach the end of life with a given usage pattern.

4 EVALUATION

In this section, we describe the details of the platform, workload scenario, evaluation metrics, and baseline algorithms. Then, we demonstrate the effectiveness of our framework against state-of-the-art algorithms in terms of QoE and BCL.

4.1 Experimental Setup

4.1.1 Platform. For evaluation purposes, we use an ODROID XU3 board, containing an Heterogeneous Multi-Processor (HMP), which is used in Samsung Galaxy series smartphones. The multi-processor consists of four "LITTLE" Cortex-A7 cores (operating in the frequency range of 200 to 1,400 MHz) and four "big" Cortex-A15 cores (operating in the frequency range of 1,400 to 2,000 MHz). Such multi-processors are used in most recent high-end smartphones such as Apple a13 bionic, Qualcomm Snapdragon 865, and Samsung Exynos 990 [28] with different numbers of LITTLE and big cores for power saving and high performance. The platform supports Dynamic Voltage and Frequency Scaling (DVFS) for each cluster using CPU-freq driver. Besides, the board provides per cluster power monitor, using an INA231 sensor [14]. Application mapping, thread-to-core binding, and task migration support are enabled through Linux system utilities. The proposed resource management framework is implemented as a Linux user-space daemon and invoked every parametrizable epoch. For experimentation purposes, we set the resource management epoch to 0.5 s, and for the plug-in/out prediction to 2 s for high accuracy; however, we can increase the plug-in/out prediction period to decrease the overhead. We use a battery model (4 V, 2,000 mAh) and allocate effective battery energy (28,800 J) to compare various approaches.

4.1.2 Workload. We select a set of applications from Mibench benchmark suites combining with web surfing as well as playing audio and video, which represent the behavior frequently encountered in heterogeneous embedded systems, in particular smartphones [7, 8, 42]. For web surfing, we used `x-www-browser` and `wget` commands, which are used in Linux command line for opening a website in the default browser and downloading a file from a specified link. For playing mp3 and mp4 files as audio and video in Linux command line, we use `play` and `mplayer` command, respectively. We open several websites while the Mibench applications are running and play 4-min audio and video files. The Mibench benchmark suites provide benchmarks in various categories of standard applications ranging from sensor systems on simple microcontrollers to smartphones and desktops [15]. We summarize the applications that we used from Mibench in this work in Table 3. The applications that we use in this work are in the network and security categories, both of which are relevant to and representative of the applications run on smartphones, in addition to automotive control applications for a wider range of applications beyond current usage in smartphones. To create a high intensive and real workload scenario, multiple applications from Table 3 run simultaneously while the user is web surfing, and video or audio are playing in the

Table 3. Summary of Mibench Applications Which Are Used in Our Experiments

Application	Category	Summary
Dijkstra	Network	Constructs a large graph and then calculates the shortest path between every pair of nodes
Patritia	Network	Creates data structure for representing routing tables in network applications
Sha	Security	Secure hash algorithm
Rijndael	Security	An advanced and standard encryption and decryption method
Qsort	Automotive Control	Sort a large array of strings into ascending

Table 4. Users' Cell Phone Models and Battery Capacity

User	Smartphone model	Battery capacity
User1,7	Samsung J7	3,300 mAh
User2,4	Fairphone1	2,000 mAh
User5	Samsung A70	4,500 mAh
User6	Samsung S8	3,000 mAh
User8	Sony Xperia XZs	2,900 mAh

background. The applications enter and leave the system dynamically and in an unknown sequence. We enhance the applications, using a Heartbeat API [18] to monitor the performance at run-time. This API provides an application-level performance metric in terms of heartbeat per epoch, which is used in several works for performance monitoring [25, 26, 39, 49]. Each application determines one performance reference based on its requirements, e.g., the required number of heartbeats per second and issues a heartbeat after every full run.

4.1.3 Methodology. We run eight different user event patterns that are extracted from real user data patterns (collected using the *Battery Log* app [19]). The event patterns contain plug-in/out events based on the time and SOC. The users used various models of cell phone with different battery capacity, which are listed in Table 4. Figure 9 shows the real SOC patterns of the eight users over a period of 4 months for six users and 2 weeks for two other users. The real SOC patterns of these eight users are used for SOC pattern generation in this article. Figure 9 shows different users can have different habits in plug-in/out. For better demonstration, we zoomed in the SOC pattern of User1 and User2. The collected data for User5 and User6 represent a shorter period of time, i.e., 2 weeks, compared to the others. This enables us to assess the effect of our method when little training data is available. The evaluation results show that even with 2 weeks of data collection, our approach can result in acceptable improvement in BCL and QoE.

We use the collected data to design a probabilistic model for each user. For evaluation purposes, we generate SOC patterns by using the probabilistic model and the battery model, which are described in Section 3.1. The applications enter the system dynamically during a day. In our experiments, the user and SOC patterns are generated for a period of 200 h.

4.2 Comparison

4.2.1 Comparison Metrics. We compare the SOC pattern, temperature of the battery, BCL, and QoE in our framework against the state-of-the-art approaches. We know that higher temperature, average SOC, and SOC swing lead to a higher aging effect and lower BCL. For evaluation purposes, we model the BCL (explained in Section 3.4) using battery temperature and SOC pattern, and we compare the normalized BCL by using various frameworks and users. The BCL is normalized

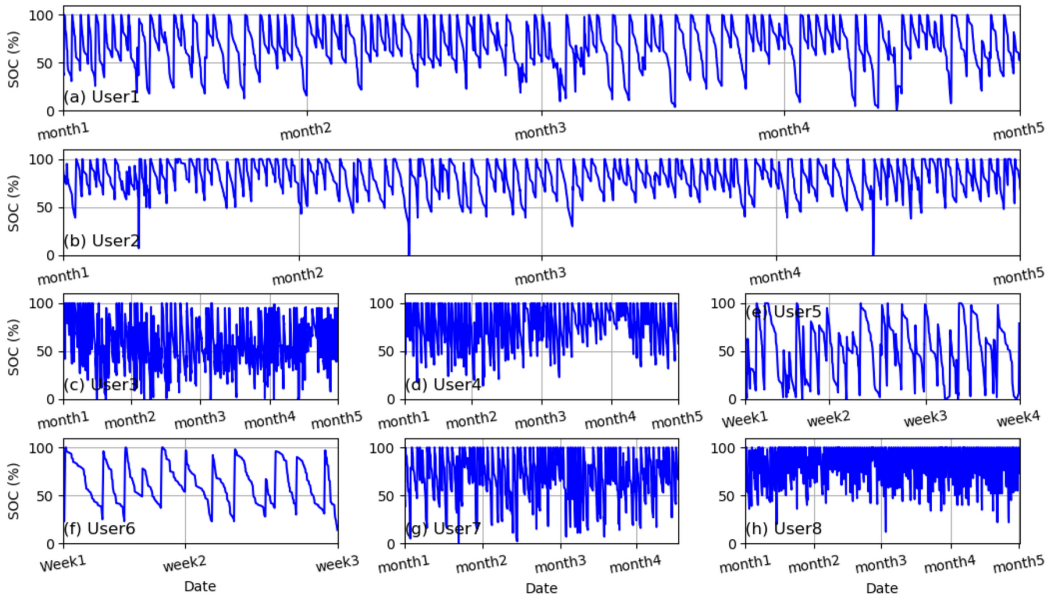

Fig. 9. Real SOC patterns of eight users over various periods of time. We use them in this work for SOC pattern generation.

to 1, using the approach with the highest BCL. For the evaluation of QoE, we measured power consumption (for energy calculation), and performance of each running application. We compare *average power/average performance* for each framework to evaluate which one performs better in terms of power-performance. By using power and performance as explained in Equation (11), we calculate the QoE. Higher QoE and BCL is better due to higher user satisfaction. However, these two objectives are conflicting, and maximizing of one of them may lead to minimizing the other. Thus, for evaluation, combination of QoE and BCL can be suitable as Figure of Merit (FOM). Although QoE is a short-term evaluation metric and BCL is a long-term one, we consider the average of QoE in long term and propose Figure of Merit (FoM) = QoE × BCL. Hence, the higher overall FoM shows relatively higher QoE and BCL. Considering QoE and BCL are both normalized in range of 0 and 1, the maximum value for FoM is also 1. In this article, we set QoE and BCL to the same weight in the evaluation, while in the future, we can combine these two factors using different weights.

4.2.2 Existing Methods. In this article, we compare our method against various resource management approaches that focus on optimizing different objectives. Table 5 categorize the methods based on the objectives that they consider for resource management. Our proposed approach in the last line of Table 5 presents a more comprehensive method compared to the others, addressing all the objectives that are mentioned in the table. In future work, we will compare our method against the approaches focusing on optimizing BCL.

1. High performance (HP): This approach focuses on maximizing the performance of applications and allocates the big cores with high frequency to the applications. This resource management approach is implemented based on the Linux Performance governor, which is available and used in off-the-shelf smartphones [36, 44].

2. Low power (LP): This approach focuses on low-power consumption and allocates the LITTLE cores with low frequency to the applications to save power. This resource management approach is

Table 5. Summary of Existing Works Against the Proposed Method

Technique	Power	Perf.	Energy	QoE	Battery	Plug-in/out pattern	Rate capacity	Temp.	BCL
HP	×	✓	×	×	×	×	×	×	×
LP	✓	×	×	×	×	×	×	×	×
SC [34]	×	×	×	×	✓	✓	×	×	✓
Od	✓	✓	×	×	×	×	×	×	×
DyPO [14]	✓	✓	✓	×	×	×	×	×	×
BUQS [22]	✓	✓	✓	✓	✓	×	×	×	×
UC [41]	✓	✓	✓	✓	✓	✓	×	×	×
Proposed method	✓	✓	✓	✓	✓	✓	✓	✓	✓

implemented based on the Linux Powersave governor, which is available and used in off-the-shelf smartphones [36, 44].

3. On-demand (Od): On-demand is a dynamic in-kernel cpufreq governor that scales the frequency according to the current workload. This resource management approach is implemented based on the Linux On-demand governor, which is available and used in off-the-shelf smartphones [36, 44].

4. Low energy (DyPO) [14]: This low-energy approach measures the energy consumption of each application for each resource management configuration, offline, then, allocates the configuration with the lowest energy consumption to the applications.

5. QoE-aware (BUQS) [22]: This method gradually decreases QoS, using DVFS and task migration based on the difference between current battery state and a battery reference. When the SOC is lower than the defined battery reference, the BUQS compromise QoS to achieve higher energy saving. The battery reference is obtained from the energy usage history of each individual user. As shown in Table 5, this method is more similar to our work compare to the others. This method uses the Odroid XU3 platform for evaluation, which is also used in our work.

6. User-centric QoE-aware (UC) [41]: This method considers user preference at run-time and sets the frequency and core allocation based on the user preference to maximize QoE. The resource management adjusts the actuation knobs for each individual user based on the plug-in prediction. This method has the most similarity with our work, as shown in Table 5.

7. Aging-aware (SC) [34]: This approach focus on decreasing the aging of smartphones batteries and does not use resource management. For decreasing the aging effect, they provide a smart charging method to minimize the average SOC by considering the user's plug-in duration and plug-out event. In this method, two approaches are used for decreasing the average SOC. The first one is delaying the charging based on plug-out prediction, and the other one is decreasing the charging limit (e.g., from 100% to 80%).

4.2.3 Summary of Proposed Methods. Here, to facilitate positioning our proposed methods compared to existing ones, we present a summarized overview of the proposed methods.

UBAR is a User- and Battery-aware Resource management approach that optimizes both QoE and BCL by considering all the objectives that are mentioned in Table 5. The rate capacity effect and temperature evaluation are considered in our battery model for more accurate results.

UBAR+ is an improved version of our framework, which combines UBAR and smart charging (SC) [34] to increase the BCL. UBAR+ set a limitation for charging (e.g., 80%) to decrease the average SOC based on *SC* method.

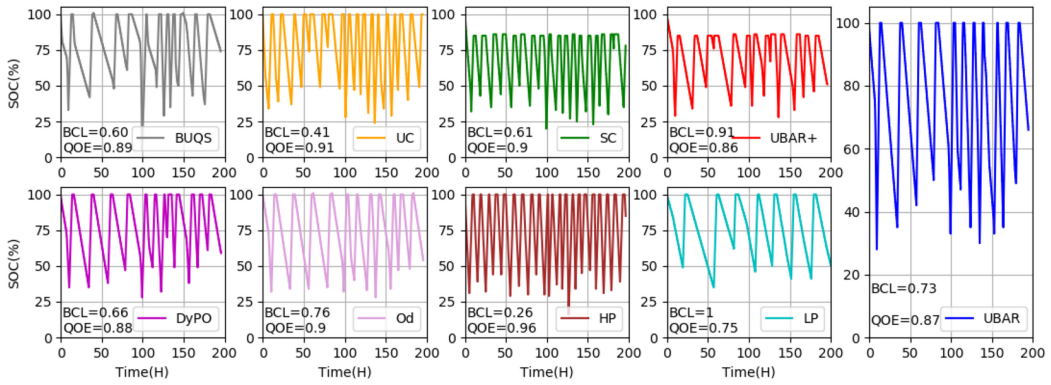

Fig. 10. 200 h of SOC patterns using nine different resource management approaches.

4.3 Evaluation of Results

Figure 10 shows the created SOC patterns during 200 h, using the nine approaches that are explained in Section 4.2. The calculated BCL and average QoE for each approach are mentioned in the subplots. The BCL is normalized to 1, using the approach with the highest BCL (e.g., *LP*). Figure 10 shows the variation of SOC swing, average SOC, rate of discharge, and charging limit by using different methods. The resource management approach that leads to the highest BCL, i.e., *LP*, has a lower rate of discharging, SOC swing and average SOC. In contrast, the *HP* approach has the highest rate of discharging among all the tested methods, which leads to the higher SOC swing and lower BCL. Such a resource management technique has relatively higher QoE due to using high frequencies to satisfy the user request on running applications. However, *LP* has the lowest QoE because of saving power without considering performance requirements. The *Od* approach that considers workload scenario and scales frequency based on that has higher rate of discharging compared to the *LP* and lower compared to the *HP*. The *Od* approach is a more balanced resource management, which results in lower BCL and higher QoE than *LP*.

Dypo, which considers both power and performance, has an SOC pattern with higher SOC swing compared to *LP* and lower swing compared to *HP*. Thus, the BCL for *Dypo* is higher than *HP*, and the QoE is higher than *LP*. *UC* and *BUQS* are two approaches that focus on increasing QoE. *UC* has relatively high QoE, which leads to high SOC swing and low BCL. *BUQS* has lower SOC swing compared to *UC*, thus the BCL for this approach is higher than *UC*. The *SC* focuses on decreasing the aging effect during charging by delaying the charging process and setting a target charge limit, however without using customized resource management, which consider both QoE and BCL, the BCL is still relatively low. The resource management that is used for *SC* is the Linux default resource management, which results in relatively high QoE. In *UBAR*, we have suitable trade-off between BCL and QoE, and in *UBAR+*, we find the best trade-off. The BCL for *UBAR+* is higher than the other approaches except *LP* (which has the lowest QoE), while the QoE is relatively high.

In summary, Figure 10 shows the approaches with higher SOC swing and rate of discharging have lower BCL, and the approaches that use limited charging can improve the BCL. As shown in Figure 9 the SOC pattern for each different user can be different. To show the impact of user behavior on the effectiveness of the approaches, we compare various methods for eight users in the following. Figures 11 and 12 show such comparison based on different metrics.

Figure 11(a) shows the comparison of *average power/average performance* during the experiments. The lower power/performance is relatively better, because it shows higher performance

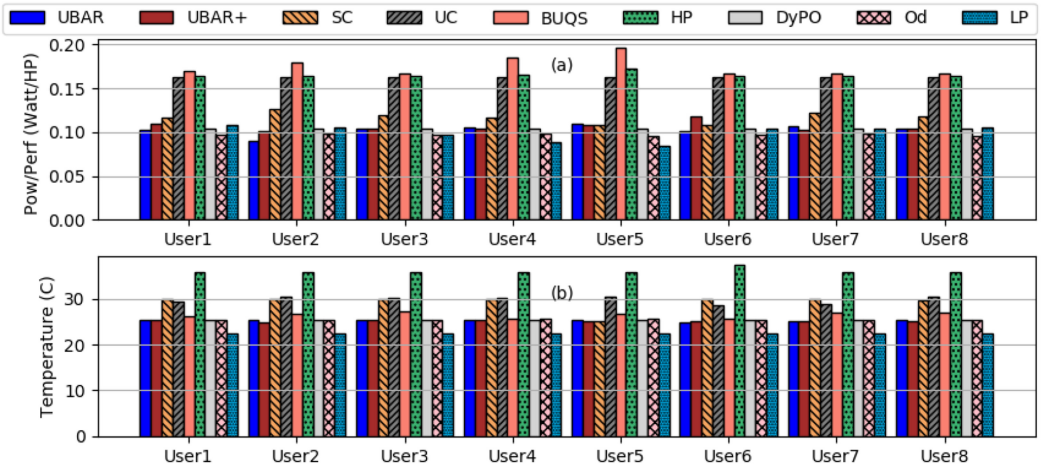

Fig. 11. Comparison of (a) Power/Performance and (b) temperature for nine different resource management approaches (lower is better). Performance is measured in Heartbeats per second.

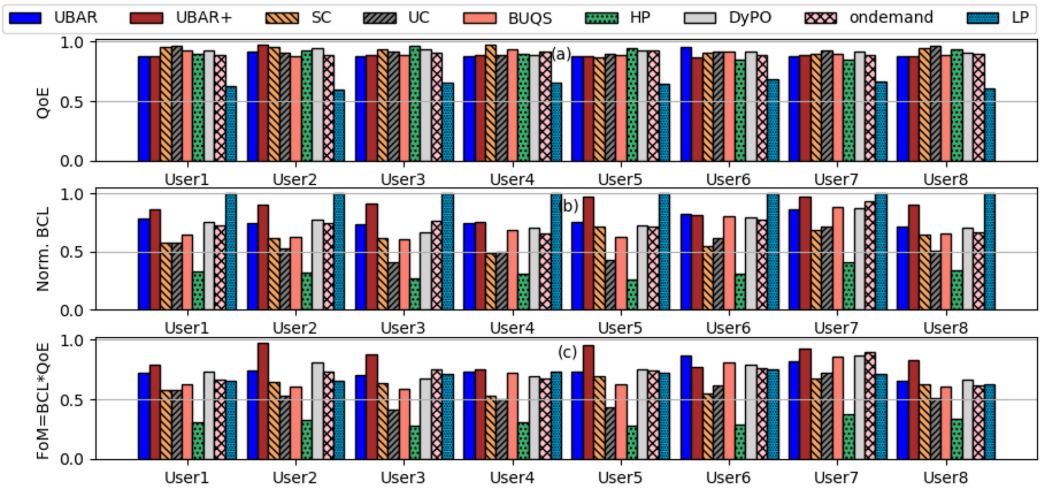

Fig. 12. Comparison of (a) QoE, (b) BCL, and (c) FoM = BCL × QoE for eight users, using nine different resource management approaches.

and lower power consumption. Figure 11(a) shows that UBAR and UBAR+ have a relatively lower power/performance for all eight users. The *LP* approach has also low power/performance, however, this method provides unacceptable absolute performance and QoE (as shown in Figure 12). *UC*, *BUQS*, and *HP* have the highest power/performance due to the high power consumption of the systems using such approaches. Figure 11(a) shows the variation in user behavior does not affect the power/performance metric dramatically, and the trend of various resource management approaches for different users is similar. Figure 11(b) shows the average battery temperature (over the whole duration of the experiment) when using each approach by each user. UBAR and UBAR+ have a relatively lower temperature, which leads to higher BCLs.

In Figure 12, we compare the (a) QoE, (b) normalized BCL, and (c) FoM = QoE × BCL for eight users, when using the nine resource management approaches. As shown in the Figure 12, *UBAR*

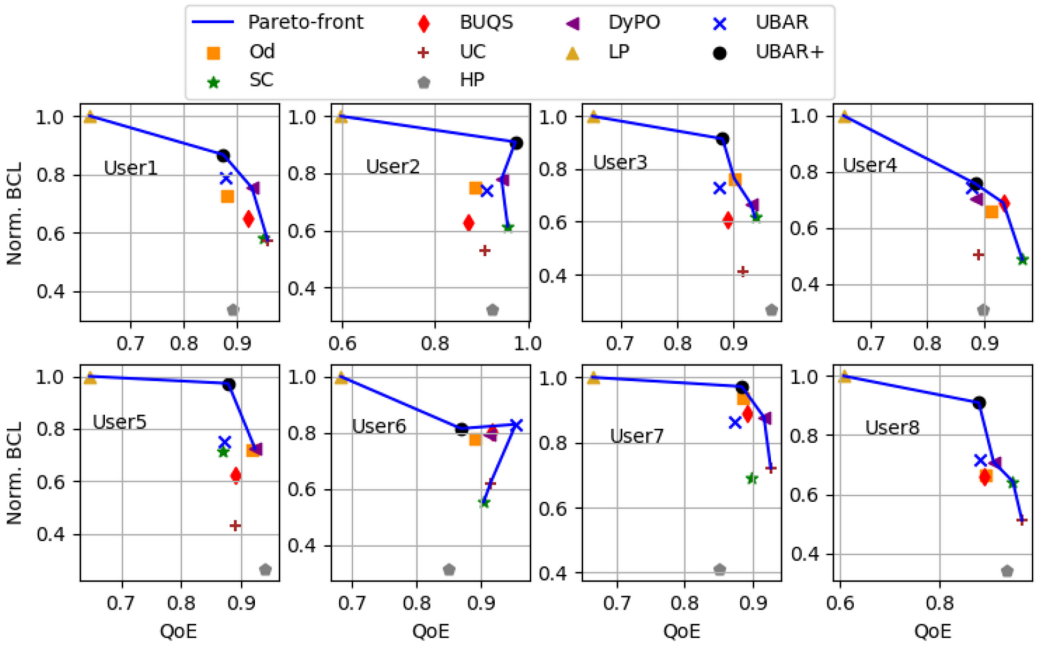

Fig. 13. Comparison of various resource management approaches for eight users in terms of QoE-BCL.

and *UBAR+* have relatively higher QoE for User2 and User6, and for the other users with a small difference in QoE have significantly higher BCLs. The *LP* that has the highest BCL, has very low QoE, which is not acceptable for users. Figure 12(c) presents multiplication of QoE and BCL, to show which approach is better. The *UBAR* and *UBAR+* have the highest value for FoM, which shows these two methods find the best trade-off for QoE and BCL. As shown in Figure 12(c), for some users, i.e., User4, User6, and User7, the FoM of the other approaches may be similar to *UBAR* or *UBAR+*, which shows the efficiency of the resource management approaches depend on users' plug-in/out pattern. In the SOC pattern of User4 and User7 (shown in Figure 9), the pattern changes significantly, which influences our prediction method. Similarly, the relatively lower amount of collected data for User6 may decrease the accuracy of the prediction method and affect the efficiency of the resource management. However, in User5, although the collected data is for 2 weeks, the FoM of our approach is the highest, which shows the plug-in/out patterns for User5 are predictable even by a small amount of data collection. Thus, if the user SOC pattern varies dramatically, then the *UBAR* and *UBAR+* may lead to the same efficiency as other approaches. Moreover, this specific instance shows that *UBAR* may result in a better value for FoM compared to *UBAR+* based on the user habits. As Figure 12(c) shows, our method compared to the other methods is most reliable, always providing the best FoM.

Figure 13 visualizes the trade-off between QoE and BCL and where each approach stands on this trade-off. The ideal solution would be at the top right corner of these plots. Therefore, the closer each approach to the top right corner, the better its overall utility. The approaches that come to lie on the Pareto-front (blue line) form the set of Pareto-optimal solutions can be considered as approaches to provide the best trade-off of QoE and BCL among the investigated strategies. The black circle and blue cross show our proposed techniques, *UBAR+* and *UBAR*. For all the users, *UBAR+* is part of the Pareto-front and *UBAR* is either on the Pareto front or very close. Figure 13 shows, that LP compromises QoE to have the highest BCL, while *HP* compromises BCL to have

Table 6. FoM Improvement of UBAR and UBAR+ Compared to Other Approached

Method	HP	UC	Od	LP	SC	BUQS	DyPO
UBAR benefit (FoM)	40%	30%	28%	10%	19%	14%	7%
UBAR+ benefit (FoM)	50%	46%	39%	29%	24%	21%	17%

high QoE. However, *UBAR* and *UBAR+* result in a good trade-off with relatively high QoE as well as high BCL. By considering the Pareto front, if the user requires high performance, *SC* and *UC* can be better solutions compared to *HP*. The other approaches may be in the Pareto front only for some users. However, our approach shows a stable superiority for all users.

In this article, we presented a resource management approach in combination with a new control knob, which is smart charging to increase the BCL, while satisfying user experience at the same time. Table 6 shows the maximum improvements that can be achieved using UBAR and UBAR+ compared to other approaches. UBAR+ achieves the highest gains compared to the methods that focus on performance improvement (e.g., *HP*) and neglect improving BCL. Figures 11, 12, and 13 show same approaches and workload may lead to various improvements for different users. The variation in user plug-in/out event causes such effects. For example, as shown in Figure 9, User2 plugs-in the device when the SOC is more than 50%, and the SOC pattern is more predictable for this user. However, the variation of SOC pattern is higher for User3 and User4. Thus, the advantage of UBAR and UBAR+ is higher for User1 and User2.

We have also evaluated the stability of our framework through extensive testing. Our resource management framework is evoked every cycle with a tunable length, which we set to 0.5 s in our experiment. The maximum clock cycle of the processor is 0.005 μs; thus, our resource management cycle is at least 100 million times larger than the operating processor cycle. Given the large differences in the two cycles, changes in the resource management scheme are considered to be slow smooth changes and will not cause an instability problem. The stability of the system is an important concern. While there are standard techniques known in dynamical systems theory to establish such stability, but they require a formal model of the system. Deriving such a model in our case is not possible, and so the stability can only be shown empirically. In the future, we plan to not only rely on extensive testing of our algorithm but to formally prove stability.

4.4 Scalability, Accuracy and Overhead

In this section, we analyze the scalability of our framework with respect to the different system parameters, and we report the overhead of UBAR and UBAR+.

4.4.1 Scalability. For analyzing the scalability of our framework, we consider the following three parameters, which can vary in a processor:

- **Number of cores:** Although the number of cores in the currently best smartphone processors is not higher than eight [28], we analyze the complexity of our method with increasing number of cores to show the scalability of our approach. By increasing the number of cores, the complexity of our framework may increase in two ways. The first one is power measurement and temperature estimation. In the systems with power or temperature sensors, the outputs of these sensors are used directly in resource management and the models for estimation are not required, thus, increasing the number of cores does not affect the complexity of our method in such systems. The Odroid Xu3 that we use in this work contains sensors for the CPU's power and temperature measurement. For the devices without sensor, we can use power models presented in References [21, 30, 37]. The complexity of such

power models increases linearly by increasing the number of cores. The second issue for scalability is storing the status of cores for resource management to find free cores when a new application arrives. We require a data structure that shows which core is free and which one is busy at run-time. For each core, we require one bit, thus the memory space that we need increases linearly with increasing the number of cores.

- **Number of clusters in a processor:** Our framework is designed for smartphones, and the current best smartphone processors contain two clusters [28], which we consider in this work. A higher number of clusters is not used in the current smartphones, thus this is out of the scope of this work. However, by increasing the number of clusters, the actuation knobs increase, which is explained in the next paragraph. We evaluate our framework in a heterogeneous processor with two clusters, but our framework is also suitable for homogeneous processors with lower complexity and actuation knobs. In the processors with one type of core, the flexibility and efficiency of run-time resource management generally decrease. Therefore, most of the recent smartphones use heterogeneous processors to handle the complexity of applications and user requirements.
- **Number of actuation knobs:** In this work, we consider two actuation knobs, which are DVFS and TM. In processors with one cluster, TM is not available, and the number of actuation knobs decreases. Thus, while the efficiency of our framework may decrease because of the nature of the considered processor, it is still a proper solution for such devices. By increasing the number of actuation knobs, the decision making becomes more complex, and we need a learning model or another heuristic model for selecting the best action at run-time. Furthermore, a system must at least have one of the above actuation knobs for being compatible with our framework.

Considering the above parameters, our framework is scalable and compatible with available heterogeneous and homogeneous systems with the mentioned limitations. In a system with more actuation knobs, we can still use just the two knobs proposed in this work, but for higher efficiency, the new knobs should be added to the system, and a new decision making algorithm, which considers all the knobs, needs to be found. While the prediction accuracy varies based on the users' habits, we try to increase the accuracy of our prediction by updating the model at run-time. We estimate the accuracy of our plug-in/out prediction models by comparing the prediction result to the real plug-in/out event for 100 cycles of charging and discharging for each user. We achieve up to 87% accuracy for plug-in prediction and 88% for plug-out prediction. We also estimate the accuracy of our temperature model by comparing it with the real temperature values, which are collected by *Battery log app* in real smartphones. The results show average and variance of temperature by using our model are 28.7 and 14 while these two parameters are 29.8 and 11 in the real temperature log. Thus the model error on average is 3.6%, which is negligible in our use case.

4.4.2 Overhead. We estimate the overhead of our approach by monitoring the CPU and memory usage with and without our framework, using the *htop* command in Linux. This command allows to interactively monitor the system's vital resources in real time. The CPU usage of UBAR is 2.6% of one of the big cores from four big and four LITTLE cores in Odroid XU3. By considering the utilization of LITTLE cores as half of one of the big cores, we can assume two big cores instead of four LITTLE cores for overhead calculation. Thus, if our framework uses 2.6% of one of the big cores, it uses 0.43% of all the cores (assuming an equivalent of 6 big cores for the entire system). Therefore, the CPU overhead of our framework is 0.43% of all the cores. The memory overhead, which is also monitored by htop, is 0.3%. Thus, with negligible overhead, UBAR provides up to 40% (as presented in Table 6) improvement compared to the state-of-the-art approaches.

5 CONCLUSIONS

In this article, we presented a UBAR framework for smartphones. UBAR considers the conflicts between power, performance, and battery temperature and finds the best trade-off between QoE and BCL. The QoE is affected by user preference at run-time, which varies based on SOC level at any given time and plug-in/out event. An efficient resource management approach must consider users' preferences variation for QoE optimization, thus, UBAR monitors the history of each user to predict SOC, plug-in/out patterns, and users' preferences, consequently. By using this information, UBAR improves the balance between performance and power, and thus optimizes QoE and BCL. We compare the UBAR framework against 9 various resource management approaches for eight different users. The evaluation results show that UBAR and UBAR+ provide up to 40% and 50% improvement compared to the existing state-of-the-art approaches while creating a meager 0.43% CPU and 0.3% memory overhead, which is negligible compared to the gained advantages. In the future, we will evaluate our approach by considering more users, and we will study different weights for QoE and BCL based on user preference and direct the resource management to maximize the weighted combination of QoE and BCL.

REFERENCES

[1] Saeid Bashash, Scott J. Moura, Joel C. Forman, and Hosam K. Fathy. 2011. Plug-in hybrid electric vehicle charge pattern optimization for energy cost and battery longevity. *J. Power Sources* 196, 1 (2011), 541–549.

[2] Alberto Bocca, Alessandro Sassone, Alberto Macii, Enrico Macii, and Massimo Poncino. 2015. An aging-aware battery charge scheme for mobile devices exploiting plug-in time patterns. In *Proceedings of the 33rd IEEE International Conference on Computer Design (ICCD'15)*. IEEE, 407–410.

[3] Aaron Carroll, Gernot Heiser, et al. 2010. An analysis of power consumption in a smartphone. In *Proceedings of the USENIX Annual Technical Conference*, Vol. 14. Boston, MA, 21–21.

[4] Jian Chen, Lizy Kurian John, and Dimitris Kaseridis. 2011. Modeling program resource demand using inherent program characteristics. *ACM SIGMETRICS Perform. Eval. Rev.* 39, 1 (2011), 1–12.

[5] Min Chen and Gabriel A. Rincon-Mora. 2006. Accurate electrical battery model capable of predicting runtime and IV performance. *IEEE Trans. Energy Conv.* 21, 2 (2006), 504–511.

[6] Yukai Chen, Alberto Bocca, Alberto Macii, Enrico Macii, and Massimo Poncino. 2016. A li-ion battery charge protocol with optimal aging-quality of service trade-off. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 40–45.

[7] Alexei Colin, Arvind Kandhalu, and Ragunathan Rajkumar. 2014. Energy-efficient allocation of real-time applications onto heterogeneous processors. In *Proceedings of the IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 1–10.

[8] Sidartha Azevedo Lobo De Carvalho, Daniel Carvalho Da Cunha, and Abel Guilhermino Da Silva-Filho. 2017. Autonomous power management for embedded systems using a non-linear power predictor. In *Proceedings of the Euromicro Conference on Digital System Design (DSD'17)*. IEEE, 22–29.

[9] Shin Donghwa, Kitae Kim, Naehyuck Chang, Woojoo Lee, Yanzhi Wang, Qing Xie, and Massoud Pedram. 2013. Online estimation of the remaining energy capacity in mobile systems considering system-wide power consumption and battery characteristics. In *Proceedings of the 18th Asia and South Pacific Design Automation Conference (ASP-DAC'13)*. IEEE, 59–64.

[10] Denzil Ferreira, Anind K. Dey, and Vassilis Kostakos. 2011. Understanding human-smartphone concerns: A study of battery life. In *Proceedings of the International Conference on Pervasive Computing*. Springer, 19–33.

[11] Eibe Frank, Mark Hall, and Bernhard Pfahringer. 2002. Locally weighted naive bayes. In *Proceedings of the 19th conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers, 249–256.

[12] Benjamin Gaudette, Carole-Jean Wu, and Sarma Vrudhula. 2016. Improving smartphone user experience by balancing performance and energy with probabilistic QoS guarantee. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'16)*. IEEE, 52–63.

[13] Ujjwal Gupta, Manoj Babu, Raid Ayoub, Michael Kishinevsky, Francesco Paterna, and Umit Y. Ogras. 2018. STAFF: Online learning with stabilized adaptive forgetting factor and feature selection algorithm. In *Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference (DAC'18)*. IEEE, 1–6.

[14] Ujjwal Gupta, Chetan Arvind Patil, Ganapati Bhat, Prabhat Mishra, and Umit Y. Ogras. 2017. Dypo: Dynamic pareto-optimal configuration selection for heterogeneous mpsocs. *ACM Trans. Embed. Comput. Syst.* 16, 5s (2017), 1–20.

[15] Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, and Richard B. Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the 4th Annual IEEE International Workshop on Workload Characterization (WWC'01)*. IEEE, 3–14.
[16] Hardkernel. 2019. ODROID-XU. Retrieved from https://www.hardkernel.com/.
[17] Liang He, Eugene Kim, Kang G. Shin, Guozhu Meng, and Tian He. 2017. Battery state-of-health estimation for mobile devices. In *Proceedings of the 8th International Conference on Cyber-Physical Systems*. 51–60.
[18] Henry Hoffmann, Jonathan Eastep, Marco D. Santambrogio, Jason E. Miller, and Anant Agarwal. 2010. Application heartbeats: A generic interface for specifying program performance and goals in autonomous computing environments. In *Proceedings of the 7th International Conference on Autonomic Computing*. 79–88.
[19] Tae-Rok Hwang. 2013. Battery Log, Version 2.0.3. Retrieved from https://play.google.com.
[20] Anil Kanduri, Mohammad-Hashem Haghbayan, Amir M. Rahmani, Pasi Liljeberg, Axel Jantsch, Nikil Dutt, and Hannu Tenhunen. 2016. Approximation knob: Power capping meets energy efficiency. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'16)*. IEEE, 1–8.
[21] Anil Kanduri, Antonio Miele, Amir M. Rahmani, Pasi Liljeberg, Cristiana Bolchini, and Nikil Dutt. 2018. Approximation-aware coordinated power/performance management for heterogeneous multi-cores. In *Proceedings of the 55th Annual Design Automation Conference*. 1–6.
[22] Wooseok Lee, Reena Panda, Dam Sunwoo, Jose Joao, Andreas Gerstlauer, and Lizy K. John. 2018. BUQS: Battery- and user-aware QoS scaling for interactive mobile devices. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference (ASP-DAC'18)*. IEEE, 64–69.
[23] Naoki Matsumura, Nobuhiro Otani, and Kiyohiro Hamaji. 2009. Intelligent battery charging rate management. U.S. Patent App. 12/059,967.
[24] Alan Millner. 2010. Modeling lithium ion battery degradation in electric vehicles. In *Proceedings of the IEEE Conference on Innovative Technologies for an Efficient and Reliable Electricity Supply*. IEEE, 349–356.
[25] Nikita Mishra, Connor Imes, John D. Lafferty, and Henry Hoffmann. 2018. CALOREE: Learning control for predictable latency and low energy. *ACM SIGPLAN Notices* 53, 2 (2018), 184–198.
[26] Nikita Mishra, Huazhe Zhang, John D. Lafferty, and Henry Hoffmann. 2015. A probabilistic graphical model-based approach for minimizing energy under performance constraints. *ACM SIGARCH Comput. Architect. News* 43, 1 (2015), 267–281.
[27] Thannirmalai Somu Muthukaruppan, Mihai Pricopi, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. 2013. Hierarchical power management for asymmetric multi-core in dark silicon era. In *Proceedings of the 50th ACM/EDAC/IEEE Design Automation Conference (DAC'13)*. IEEE, 1–9.
[28] Myfixguide. 2020. Best Smartphone Processors Ranking. Retrieved from https://www.myfixguide.com/best-smartphone-processors-ranking/.
[29] Gang Ning and Branko N. Popov. 2004. Cycle life modeling of lithium-ion batteries. *J. Electrochem. Soc.* 151, 10 (2004), A1584.
[30] Anuj Pathania, Qing Jiao, Alok Prakash, and Tulika Mitra. 2014. Integrated CPU-GPU power management for 3D mobile games. In *Proceedings of the 51st ACM/EDAC/IEEE Design Automation Conference (DAC'14)*. IEEE, 1–6.
[31] Tina R. Patil. 2013. Performance analysis of Naive Bayes and J48 classification algorithm for data classification. *J. Comput. Sci. Appl.* 6, 2 (2013).
[32] Matthew B. Pinson and Martin Z. Bazant. 2012. Theory of SEI formation in rechargeable batteries: Capacity fade, accelerated aging and lifetime prediction. *J. Electrochem. Soc.* 160, 2 (2012), A243.
[33] Alma Pröbstl, Bashima Islam, Shahriar Nirjon, Naehyuck Chang, and Samarjit Chakraborty. 2020. Intelligent chargers will make mobile devices live longer. *IEEE Design Test* 37, 5 (2020), 42–49.
[34] Alma Pröbstl, Philipp Kindt, Emanuel Regnath, and Samarjit Chakraborty. 2015. Smart2: Smart charging for smart phones. In *Proceedings of the IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 41–50.
[35] Amir-Mohammad Rahmani, Mohammad-Hashem Haghbayan, Anil Kanduri, Awet Yemane Weldezion, Pasi Liljeberg, Juha Plosila, Axel Jantsch, and Hannu Tenhunen. 2015. Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach. In *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED'15)*. IEEE, 219–224.
[36] Basireddy Karunakar Reddy, Geoff V. Merrett, Bashir M. Al-Hashimi, and Amit Kumar Singh. 2018. Online concurrent workload classification for multi-core energy management. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'18)*. IEEE, 621–624.
[37] Hergys Rexha, Simon Holmbacka, and Sébastien Lafond. 2017. Core level utilization for achieving energy efficiency in heterogeneous systems. In *Proceedings of the 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP'17)*. IEEE, 401–407.
[38] Leonardo M. Rodrigues, Carlos Montez, Ricardo Moraes, Paulo Portugal, and Francisco Vasques. 2017. A temperature-dependent battery model for wireless sensor networks. *Sensors* 17, 2 (2017), 422.

[39] Elham Shamsa, Anil Kanduri, Amir M. Rahmani, Pasi Liljeberg, Axel Jantsch, and Nikil Dutt. 2018. Goal formulation: Abstracting dynamic objectives for efficient on-chip resource allocation. In *Proceedings of the IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC'18)*.

[40] Elham Shamsa, Anil Kanduri, Amir M. Rahmani, Pasi Liljeberg, Axel Jantsch, and Nikil Dutt. 2019. Goal-driven autonomy for efficient on-chip resource management: Transforming objectives to goals. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'19)*. IEEE, 1397–1402.

[41] Elham Shamsa, Anil Kanduri, Nima TaheriNejad, Alma Pröbstl, Samarjit Chakraborty, Amir M. Rahmani, and Pasi Liljeberg. 2020. User-centric resource management for embedded multi-core processors. In *Proceedings of the 33rd International Conference on VLSI Design and 19th International Conference on Embedded Systems (VLSID'20)*. IEEE, 43–48.

[42] Shervin Sharifi, Dilip Krishnaswamy, and Tajana Šimunić Rosing. 2013. PROMETHEUS: A proactive method for thermal management of heterogeneous MPSoCs. *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 32, 7 (2013), 1110–1123.

[43] Yanzhi Wang, Xue Lin, Qing Xie, Naehyuck Chang, and Massoud Pedram. 2014. Minimizing state-of-health degradation in hybrid electrical energy storage systems with arbitrary source and load profiles. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'14)*. IEEE, 1–4.

[44] XDA. 2015. XDA-developersforums. Retrieved from https://forum.xda-developers.com/general/general/ref-to-date-guide-cpu-governors-o-t3048957.

[45] Qing Xie, Jaemin Kim, Yanzhi Wang, Donghwa Shin, Naehyuck Chang, and Massoud Pedram. 2013. Dynamic thermal management in mobile devices considering the thermal coupling between battery and application processor. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'13)*. IEEE, 242–247.

[46] Rui Xiong, Jiayi Cao, Quanqing Yu, Hongwen He, and Fengchun Sun. 2017. Critical review on the battery state of charge estimation methods for electric vehicles. *IEEE Access* 6 (2017), 1832–1843.

[47] Kaige Yan, Xingyao Zhang, and Xin Fu. 2015. Characterizing, modeling, and improving the QoE of mobile devices with low battery level. In *Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'15)*. IEEE, 713–724.

[48] Kaige Yan, Xingyao Zhang, Jingweijia Tan, and Xin Fu. 2016. Redefining QoS and customizing the power management policy to satisfy individual mobile users. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. IEEE, 1–12.

[49] Huazhe Zhang and Henry Hoffmann. 2016. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. *ACM SIGPLAN Notices* 51, 4 (2016), 545–559.

[50] Yancheng Zhang and Chao-Yang Wang. 2009. Cycle-life characterization of automotive lithium-ion batteries with LiNiO2 cathode. *J. Electrochem. Soc.* 156, 7 (2009), A527.

[51] Yuhao Zhu, Matthew Halpern, and Vijay Janapa Reddi. 2015. Event-based scheduling for energy-efficient qos (eqos) in mobile web applications. In *Proceedings of the IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. IEEE, 137–149.

Received April 2020; revised October 2020; accepted December 2020