# User-centric Resource Management for Embedded Multi-core Processors

Elham Shamsa[1], Anil Kanduri[1], Nima TaheriNejad[2], Alma Pröbstl[3], Samarjit Chakraborty[3],
Amir M. Rahmani[4], Pasi Liljeberg[1]
[1]*Department of Future Technologies, University of Turku, Turku, Finland*
[2]*Institute of Computer Technology, TU Wien, Vienna, Austria*
[3]*Institute of Computer Technology,Technical University of Munich, Munich, Germany*
[4]*Department of Computer Science, University of California, Irvine, USA*
{elsham, spakan, pakrli}@utu.fi, nima.taherinejad@tuwien.ac.at,
{alma.proebstl, samarjit}@tum.de, a.rahmani@uci.edu

***Abstract*—Modern battery powered Embedded Systems (ES) must provide a high performance with minimal energy consumption to enhance the user experience. However, these two are often conflicting objectives. In current ES resource management techniques, user behavior and preferences are only indirectly or not at all considered. In this paper, we present a novel user- and battery-aware resource management framework for multi-processor architectures that considers these conflicting requirements and dynamic unknown workloads at run-time to maximize user satisfaction. Proposed technique learns user's habits to dynamically adjust the resource management schemes based on the data it collects regarding user's plug-in behavior, battery charge status, and workloads variability at run-time. This information is used to improve the balance between performance and energy consumption, and thus optimize the Quality of Experience (QoE). Our evaluation results show that our framework enhances the user experience by 22% in comparison with the existing state-of-the-art.**

***Index Terms*—Resource Management, Personalization, Quality of Experience, User-awareness, Battery-awareness, Performance, Energy Consumption, Heterogeneous, Multi-processor**

## I. Introduction

Increasing usage of battery powered mobile embedded systems such as smart phones and wearable devices rises the importance of maximizing Quality of Experience (QoE) of users. In addition to graphical interface and performance of applications, QoE depends on user's satisfaction with the device in terms of battery life [1]. However, battery charging patterns vary among different users, resulting in different energy utilization dynamics and battery drain for each given user [2]. For example, some users prefer to charge their phones over night with long plug-in times while others charge their phones ad-hocly and immediately unplug once the full charge level has been reached. Battery plug-in pattern and State of Charge (SOC), i.e., the amount of battery life remaining at a given time are intertwined factors which affect the QoE. For example, consider a user who trivially prefers energy saving mode of operation at SOC=20%. However, the same user would alter the preference to high performance if the device is plugged in to a battery source, even at the same SOC of 20%. A user's preference on performance versus energy saving mode of operation depends on i) the current SOC at any given time, ii) availability of energy source (device is plugged-in/out) and iii) the likelihood of plugging in the battery source, given the current SOC. While SOC is subjective to workload characteristics of applications being run, the likelihood of battery source being plugged in is user specific.

Run-time resource management policies can improve battery life by exploiting application characteristics [3], scheduling [4], power knob actuation [5], and/or a combination of the above. Resource management policies that are designed generically do not consider user-specific parameters such as device usage history, and charge and discharging patterns [1], [6], [7], limiting their efficiency in maximizing QoE [8]. Other techniques that are user-specific do not consider battery plug-in patterns and target maximizing a generalized Quality of Service (QoS) model [9], [10]. A comprehensive method that monitors dynamic workload characteristics, current SOC and battery plug-in status, and learns the likelihood of battery being charged at an SOC can maximize QoE by adapting resource allocation decisions. To this end, we propose a user-centric resource management framework for maximizing the QoE of embedded devices' users, specifically considering personalized battery plug-in patterns and SOC.

We use Naive Bayes classifier to model the user plug-in time and predict the available energy budget to make appropriate resource allocation decisions that maximize QoE. We initially train our prediction model offline with data from different users and update the model at run-time to customize the resource management for each specific user. We use Dynamic Voltage and Frequency Scaling (DVFS) and task migration knobs to actuate performance and energy budgets. We use the real statistical usage data for simulating the user and battery state in our framework. Our contributions are as follows:

- Analysis of statistical battery data of each individual user and its effect on resource management.
- A user- and battery- aware energy optimization approach that efficiently adapts to the workload variation, individual user behavior, and performance requirements.
- A user and battery model by using online and offline learning based on real statistical data.
- A quantified model for QoE which is compatible with SOC and individual users.

- A resource management framework which customizes itself for each individual user to maximize user satisfaction.
- Evaluation of the framework on a real heterogeneous platform, namely Odroid XU3, over realistic workloads.

We provide significance of using user-centric resource management in Section II. Our framework is described in Section III. The evaluation of the resource management method over MiBench benchmark suite on Odroid XU3 HMP platform and comparison against state-of-the-art approaches is presented in Section IV. Finally, Section V concludes the paper.

## II. Significance

In this section, we motivate the importance of considering battery awareness and user-behavior in resource management and provide an overview of the state-of-the-art approaches.

### A. Motivation

To understand the significance of battery-awareness and user behavior patterns, we collected data of battery SOC and plug-in for 10 different real mobile users over 15 days. Each subplot on Figure 1 shows the probability distribution of plug-in at a certain initial SOC for a user. As shown in Figure 1, each user has a specific plug-in probability pattern, while the pattern varies among different users. For each individual user, there is at least one SOC in which the plug-in event has the highest probability. For example, User1 and User2 plug in the device more frequently in two different SOCs ($SOC = 40\%$ and $SOC = 20\%$, respectively). These different patterns result in different user expectations, which affect resource management. We demonstrate this effect through an example in Figure 2. In this example, we consider User1 and User2 in Figure 1, and customize two different resource management policies based on their preference between performance and energy. The resource management policy implicitly changes the weight of energy and performance as per the corresponding user preference, which depends on the SOC and user plug-in pattern. User1 usually plugs in the device at $SOC = 40\%$, and User2 usually plugs in the device at $SOC = 20\%$. As a test case, we use `sha` and `qsort` application from the MiBench benchmark suit [11]. The `sha` is running and the `qsort` arrives after 10s. In Figure 2 (a), the resource management is customized for User1 - thus it recognizes with a high probability that there will be a plug-in event soon. Hence, it is permissible to consume more energy to satisfy performance, which results in higher QoE for User1. The same resource management action leads to lower QoE for User2, as shown in Figure 2(a). At $t = 10s$, when the second application arrives, the resource management, which is customized for User1, causes a reduction in QoE for User2. This is due to the fact that the behavior of User1 and User2 have been historically different at that similar SOC. In Figure 2(b), when $SOC = 20\%$, User1 prefers energy saving because the SOC is lower than the expected ($SOC = 40\%$) thus resource management adapts to satisfy user expectation. The reduction at $t = 10s$ shows arrival of a new application, which has lower performance to conserve more energy. At $t = 27s$

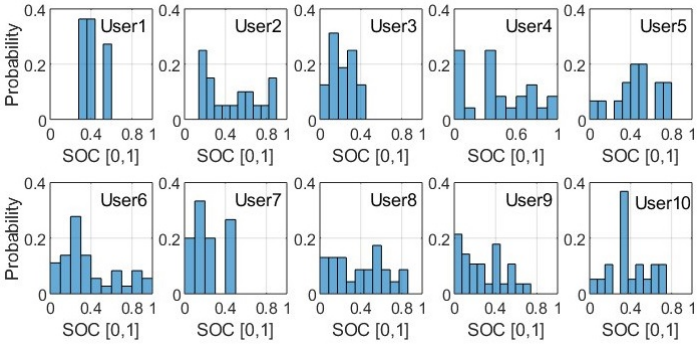


Fig. 1: Probability distribution of plug-in event in different initial SOC level for 10 different users.

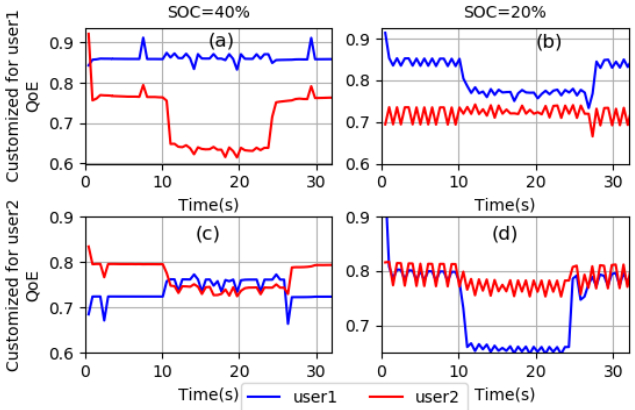


Fig. 2: QoE for two different users with two different resource management which are customized for each user.

the second application leaves the system, consequently, QoE increases again. Figure 2 (c), (d) show QoE for User1 and User2 at $SOC = 40\%$ and $SOC = 20\%$, using customized resource management for User2. They show QoE for User2 is higher than User1 thanks to using a resource management that is personalized for that user. User2 frequently plugs in their phone at $SOC = 20\%$, and expects higher performance than User1 at this SOC.

### B. Related Work

Several works on run-time resource management have been proposed to optimize performance and energy for embedded multi-core processors [12]. They use control-based model [13], offline optimization techniques [14], online machine learning [12] or a combination of these techniques [15]. However, these approaches do not consider user experience as a factor in resource management. Some works consider user experience and quantify QoE to minimize battery life and improve QoS for mobile users [9], [10], [16]. In [10], authors focus on mobile device with low SOC and characterize the QoE by capturing various users experience when the SOC is low. The approach in [16] presents the concept of energy-efficient QoS to optimize the energy under QoS constraints. In [9] an approach for optimizing energy of applications running on smartphone devices is presented that ensure a specified level of user satisfaction. These approaches consider user satisfaction, however, they use a general model for QoE and do not consider

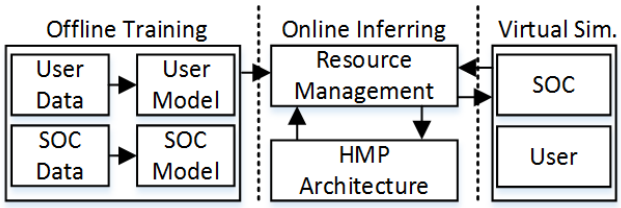


Fig. 3: Overview of the user-centric resource management framework

the user-specific data, history of the phone usage and charge and discharging behavior. These approaches quantify the QoE by averaging limited usage patterns of users and ignore the individual behavior of each user. They do not customize user-centric resource management to maximize the QoE. The resource management that is presented in [1] propose a new definition for QoE, however does not consider the plug-in behavior and battery SOC, which play an important role in changing QoE. To address the aforementioned limitations, we propose a user-centric resource management approach that considers battery SOC and user history for plug-in and plug-out to maximize the QoE.

## III. User-centric Resource Management Framework

In this section, we present our user-centric resource management framework, which considers user's battery charging patterns and SOC. Figure 3 shows the overview of the proposed approach, which is split into three phases viz., i) off-line training for SOC and user models, ii) on-line inferring for resource allocation decisions based on the trained user and SOC models, and iii) virtual user and SOC simulation for evaluation. We first build analytical models for predicting user's charging pattern - to determine the probability of plug-in/out, charging/discharging behavior of battery - to estimate current SOC. We infer to these models at run-time to make appropriate resource allocation decisions that effect performance and energy consumption of the device. To emulate a realistic battery plug-in/out and battery draining patterns, we build a virtual user and SOC agents for test cases. Our proposed approach, analytical modeling, resource allocation controller and virtual simulation environments are described in the following.

### A. User and SOC Modeling

Our resource allocation decisions depend on current SOC and user's action i.e., battery plugged-in/out. This requires i) prediction of user's action, ii) estimation of the rate at which the battery charges (upon the event that battery is actually plugged-in), and iii) estimation of the rate at which the battery discharges.

**Data Collection:** To predict the user plug-in pattern and battery charging rates, we record smartphone usage data for three different users. The data are collected over a period of 1 year for User1 and 6 months for User2 and User3 using the Battery Log app [17]. The gathered data contains a timestamp, SOC, battery temperature, voltage and the battery status such as plugged, unplugged, charging, and full. This data is split into a training and testing set, half of the collected data are used to train the user model as described in the following, and half of that is used to evaluate the experimental results.

**User Model:** User model is built to predict whether the user will plug-in the battery to a source of charging in the subsequent time stamp, given the current SOC. The user model is trained offline based on collected data and is updated online with new user data, for prediction accuracy. A fast and reliable algorithm for building a probabilistic model based on a set of data is Naive Bayes. By using Naive Bayes, the model can be also build completely on-line, because of fast convergence of this algorithm. We need to predict the probability of a user plugging-in to a source at run-time, thus using a fast algorithm such as Naive Bayes is important. The model is used to first calculate the probability of a user plugging-in to a source at run-time. Naive Bayes is a method for probabilistic classification that calculates a set of probabilities by considering the frequency of observing different values in a given set of data [18]. Bayes theorem can expressed as follows:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \tag{1}$$

where $P(A|B)$ is the probability of event $A$ happening, when event B is true (has happened). In the context of this work, we predict the probability of plug-in for different SOCs, i.e.,

$$P(A) = P(\text{Plug-in==true}), P(B) = P(\text{SOC==b}) \tag{2}$$

where $b$ is the SOC level. We use the calculated probability to generate a constrained random number, which predicts the required binary outcome on whether the user may plug-in at the next time step or not.

**SOC Charging Model:** Upon the event of a user-plug-in, we use the charging model to estimate the current SOC. A linear model based on regression analysis over the real user data collected is used for representing the rate of charging. The model is expressed as follows:

$$SOC_t = \gamma \times t + SOC_0 \tag{3}$$

where $SOC_0$ is the initial value of the SOC when the device is plugged in, $t$ is the time which is passed after plug-in (in seconds), and $\gamma$ is the regression coefficient which is set to $0.016$ in our model.

**SOC Discharging Model:** When the device is not plugged-in, we use the discharging model to estimate SOC. We monitor the instantaneous power consumption of the device over a time period to calculate the energy drained during this period.

$$E_c = P_t \times \Delta t, \tag{4}$$

where $E_c$ is the energy consumption, $P_t$ is the power consumption at the current time step and $\Delta t$ is the time duration of each time step. We calculate the current SOC of the battery by considering previous SOC and the energy drained during the current time interval, as follows:

$$SOC_t = SOC_{t-1} - \frac{E_c \times 100}{E_T}, \tag{5}$$

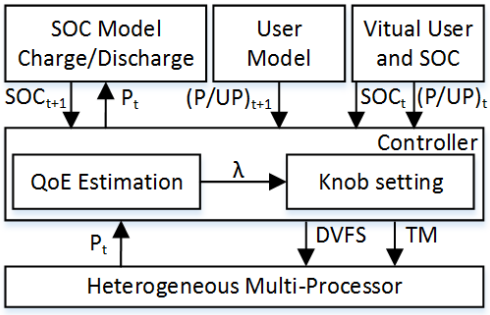


Fig. 4: High level architecture of resource management framework.

where $SOC_t$ and $SOC_{t-1}$ are SOC of the battery in current and previous time steps, $E_c$ is the consumed energy of the current time step, and $E_T$ is total energy of the battery. The battery simulated in our study has nominal capacity of $2000mAh$ and average voltage of $4V$ (i.e., $E_T = 28800J$).

### *B. Resource management*

Figure 4 shows the architecture of the resource management framework, which integrates user and battery models, current set of applications running, resource actuation knobs and the Heterogeneous Multi-Processor (HMP) system. HMP platforms have more controlling knobs than homogeneous platforms, thus the presented controller is also applicable on homogeneous systems. The controller is responsible for making resource allocation decisions that maximize QoE. The actuation settings are guided by the predicted user's action ($(P/UP)$) and estimated SOC from the user and SOC models.

**QoE Estimation:** As proposed in [1], we quantify QoE as a weighted function that combines energy and performance, expressed as follows:

$$QoE = \lambda \times Perf_N + (1-\lambda)(1-e_N). \tag{6}$$

where $\lambda$ is a user specific weight that shows user's preference between high performance and energy saving, $Perf_N$ is the average normalized performance for running applications, and $e_N$ is normalized energy consumption of the system. This model implicitly corresponds to a higher QoE with higher performance and lower energy consumption. We calculate the user's preference on performance versus energy ($\lambda$) based on the user and SOC models that are described above. While the user model predicts the current status of the battery (plug-in/out), the SOC (charging/discharging) model estimates the current SOC. $\lambda$ varies at run-time based on predicted SOC and plug-in and plug-out probabilities, which were learned for each individual user. When the user plugs in the device, $\lambda = 1$, which corresponds to the highest weight for performance since when the device is plugged in the user does not need energy saving. When the device is not plugged in, the $\lambda$ is calculated using:

$$\lambda = \alpha \times \frac{SOC}{100} \tag{7}$$

where $\alpha$ is determined at run-time based on the prediction of the plug-in event. This preference is used to determine the settings of resource actuation knobs, in order to maximize the QoE within the selected preference. We used Naive Bayes classifier to calculate the probability of plug-in, in different SOCs (described in Section III-A), and predict the plug-in event for the next cycle for each individual user. When the user model predicts a plug-in, the value of $\alpha$ increases which leads to a higher $\lambda$. Higher $\lambda$ shows higher preference of the user for performance, rather than energy. In our framework, when there is not any prediction for a plug-in event, $\alpha = 1$. When the user model predicts a plug-in for a user, $\alpha$ is experimentally set to 1.6 to reflect a higher performance priority.

**Knob Settings:** We assume an HMP as the baseline where DVFS and task migration between different types of cores (big - high performance/LITTLE - energy efficient) can be used to scale up/down performance and energy consumption. The voltage and frequency settings and choice of an appropriate (set of) core(s) depends on the predicted user action and SOC, measured SOC and power consumption, and applications' requirements. The Controller unit allocates resources in order to maximize QoE by mapping new applications to proper cores and perform DVFS and task migration based on the calculated $\lambda$ at run-time. When a new application enters the system, it is mapped on a LITTLE core, if the weight of performance is less than the energy (i.e., $\lambda < 0.5$); otherwise it is mapped on a big core. During the run-time, when ($\lambda < 0.3$), the applications migrate from big cores to LITTLE cores to save more energy (applications with lower requirements migrate first), and when ($0.3 < \lambda < 0.5$) the resource management changes the frequency by DVFS actuation. The frequency is related to the $\lambda$ and is calculated as follows:

$$Freq = \lambda \times 1000 + 1000. \tag{8}$$

The frequency step for DVFS actuation in our platform is 100 MHz, thus we round the value of $Freq$ in Equation (8) to its closest quantized level, then actuate the DVFS. When ($\lambda > 0.5$), if there is any application on LITTLE cores which does not satisfy the performance requirement, the application migrates to the big core; otherwise the DVFS actuates and the frequency which is related to the $\lambda$ is set. The *Controller* unit continuously monitors the power consumption at run-time and if the power is more than the Thermal Design Threshold (TDP), the frequency is reduced by 100 MHz in order to prevent power violation.

**Virtual User and SOC:** For experimental evaluation, we design a virtual user and SOC model which interact with the resource management framework to emulate the behavior of the user and battery. Figure 3 shows these components in high level architecture of our framework. We created a virtual agent to model the user plug-in behavior, based on the real data case study of mobile users (explained in Section III-A). This agent receives the current SOC of the battery and simulates plug-in behavior of the user based on the collected statistical data of the user. By analyzing the collected data, the probability of plug-in of each user at each SOC is determined. Then, using a roulette wheel associated with the calculated probability, a random number determines whether the user plugs in at that

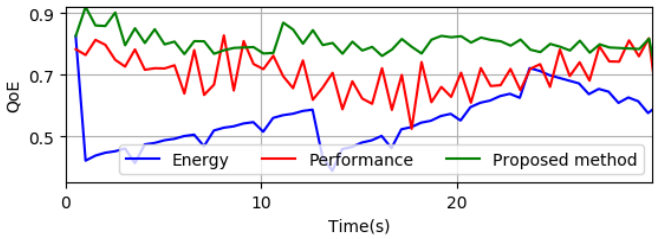


Fig. 5: Comparison of QoE for two fixed objective resource management approaches which focus on energy and performance against proposed framework.

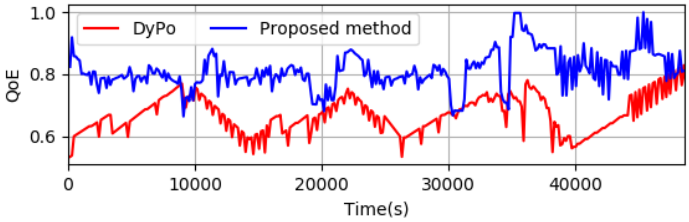


Fig. 6: Comparison of the personalized battery-aware resource management with DyPO [14] technique.

TABLE I: Comparison of proposed solution with state-of-the-art

| Technique | Energy (J) | Perf. (HB) | Time | QoE |
|---|---|---|---|---|
| DyPO [14] | 0.5 | 13.1 | 1 | 0.65 |
| User-aware | 0.87 | 17.3 | 0.89 | 0.83 |

instant or not. To model the battery, its charge level, and its changes, we use the same models presented in Section III-A.

## IV. Experimental evaluation

In this section, we describe the experimental setup, and present the evaluation of our approach against state-of-the-art resource management strategies.

### A. Experimental setup

We evaluate our framework by running applications on an Odroid XU3 board with a Samsung Exynos processor (4 big and 4 LITTLE cores) [19]. The LITTLE cores provide energy efficiency whereas the big cores deliver high performance. We measure instantaneous power consumption using on-board power sensors and integrate it over time to calculate the energy consumption. For performance measurement, we annotate each application with the Application Heartbeats API [20], to periodically log the performance in terms of heartbeats/time. Our framework is implemented as a Linux daemon on top of MARS framework [21]. The controller is invoked periodically every (parametrizable) epoch, which is set to $500ms$ in our evaluation. We use a set of applications from real MiBench benchmark suite and create an unknown and dynamic workload scenarios. These set of applications were chosen to represent the behavior frequently encountered in heterogeneous embedded systems. These contain three instances of `sha`, two instances of `qsort`, one instance of `dijkstra-large`, and `patricia` which enter and leave the system in an unknown manner. The experiments start with $SOC = 100\%$ and show at least four cycles of charging and discharging of the battery.

### B. Evaluation

We evaluate our proposed approach over 3 different baseline case studies viz., i) comparison of our approach against non-QoE aware resource management strategies, ii) comparison of our approach against non-user-centric resource management approach, and iii) evaluation of our approach for 3 different users. We use QoE as the metric to evaluate the efficiency of our approach, which is measured as described in Equation 6.

Figure 5 demonstrates the effect of using our user-centric resource management framework in comparison with two other resource management schemes which focus only on either energy or performance optimization. These two schemes are identical to our proposed approach, except that their priority on performance and/or energy are fixed manually ($\lambda$ set to 0 or 1). This limits them from satisfying the overall QoE model, in addition to the lack of user action and SOC estimation. As shown in Figure 5, the overall QoE in our proposed resource management which is adaptable to user preference is higher than two other schemes with fixed objectives. The other two schemes are efficient only in satisfying either of the two parameters among performance and energy.

Next, we compare our framework against another relevant state-of-the-art resource management approach, namely DyPO [14]. DyPO minimizes energy consumption under performance constraints by finding the optimized configuration for each application based on offline characterization. However, this approach does not consider user specific and battery state for resource management. To compare our approach against DyPO, we simulated one user's charging patterns and corresponding battery status (as described in Section III-A). Figure 6 presents the measured QoE of the user using both DyPO resource management approach and the proposed user-centric approach. At the beginning, when the SOC of the battery is high, DyPO has a low QoE. When SOC decrease during the run-time, QoE for DyPO increases which shows DyPO is a useful approach for saving energy in lower SOCs. Using DyPO, in every cycle of charging and discharging the QoE decrease and increase in a similar pattern. However, our framework adapts to the changes in the SOC of the battery and considers user plug-in behavior, which leads to relatively more persistent and higher QoE. Although QoE decreases in some points because of the variation of workload intensity, the average QoE with our approach is 0.83, which is higher than that of DyPO. Table I lists the average energy consumption in each time step, average performance metric (Heartbeat) for running applications, normalized execution time, and QoE for each approach. Table I shows our approach maximizes the QoE and provides a higher performance and lower execution time. The DyPO approach, on the other hand, minimizes energy by compromising on performance and QoE, which leads to 22% lower QoE in comparison with our framework.

For evaluating the efficiency of per-user customization of

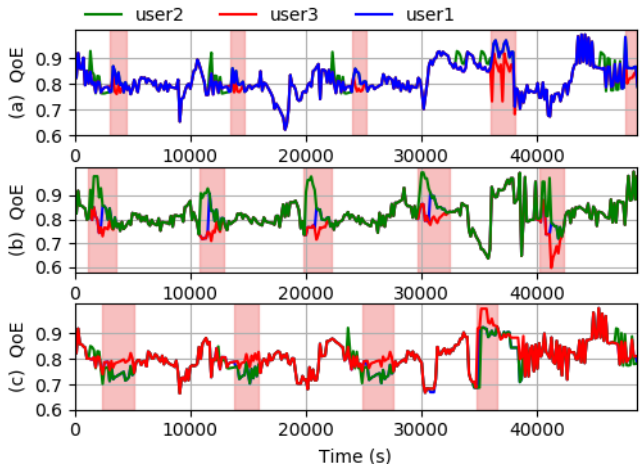


Fig. 7: Comparison of QoE for three different users. Resource Management personalized for (a) user1, (b) user2 and (c) user3.

resource management, we experimented with 3 different users with different performance and energy preferences. Our proposed resource management framework adapts to the QoE requirements of each user by customizing resource actuation knobs accordingly. Figure 7 shows the measured QoE for the three different users when running three versions of our proposed resource management, each of them customized for an individual user. The QoE remains higher than 0.8 for every user, over a larger chunk of simulation time. In some periods the QoE for one user is higher than the others. This difference of QoE for each individual user is caused due to the plug-in events which are predicted by our framework for each user. The user-centric resource management predicts the plug-in event for each user, and adjusts the resource management based on this prediction, which leads to an increase in QoE for that user. These points which are highlighted in Figure 7 show the efficiency and advantage of the proposed personalized resource management. The prediction accuracy of our framework is 83%, which is high enough to improve the QoE for each user. In the case of a wrong prediction, resource management increases the weight of performance on the predicted plug-in event. However, the user does not expect high performance at that time, thus the QoE decreases by 22% in the worst case. Table II shows the average QoE of each user during plug-in times when the resource management scheme is personalized for the user. The highlighted cells show that the highest QoE for each user is obtained when the resource management is personalized for that particular user. These highlight the advantage of customizing resource allocation per each individual user in maximizing QoE, and the significance of user action prediction and SOC estimation.

## V. Conclusion

In this paper, we proposed a user-centric resource management framework for HMP architectures. The proposed framework, learns the behavior of user and maximizes the QoE for each individual user. The proposed framework dynamically adjusts the resource management schemes based on the data

TABLE II: The average QoE of each user during plug-in time when the resource management scheme was personalized for each user.

| QoE | User1-RM | User2-RM | User3-RM |
|---|---|---|---|
| User1 | 0.85 | 0.80 | 0.77 |
| User2 | 0.8 | 0.86 | 0.77 |
| User3 | 0.79 | 0.76 | 0.82 |

it collects regarding user plug-in behavior, battery charge status, and workload variability at run-time. In particular, it uses the history of each user to learn about their personal habits and thus predict when a particular user may plug in their device. This information is then used to improve the balance between performance and energy consumption, and thus optimize the QoE. Our evaluation results show our proposed resource management improves the QoE for each individual user compared with the state-of-the-art.

## References


[1] K. Yan *et al.*, "Redefining QoS and customizing the power management policy to satisfy individual mobile users," in *Proc. of MICRO*, 2016.

[2] H. Falaki *et al.*, "Diversity in Smartphone Usage," in *Proc. of MobiSys*, 2010.

[3] A. Kanduri *et al.*, "Approximation-aware coordinated power/performance management for heterogeneous multi-cores," in *Proc. of DAC*, 2018.

[4] Q. Zhang *et al.*, "A double deep Q-learning model for energy-efficient edge scheduling," *Trans. on Services Computing*, 2018.

[5] E. Shamsa *et al.*, "Goal-Driven Autonomy for Efficient On-chip Resource Management: Transforming Objectives to Goals," in *Proc. of DATE*, 2019.

[6] X. Li *et al.*, "SmartCap: user experience-oriented power adaptation for smartphone's application processor," in *Proc. of DATE*, 2013.

[7] D. Shingari *et al.*, "DORA: optimizing smartphone energy efficiency and web browser performance under interference," in *Proc. of ISPASS*, 2018.

[8] W. Lee *et al.*, "BUQS: battery-and user-aware QoS scaling for interactive mobile devices," in *Proc. of ASPDAC*, 2018.

[9] B. Gaudette *et al.*, "Improving smartphone user experience by balancing performance and energy with probabilistic QoS guarantee," in *HPCA*, 2016, pp. 52–63.

[10] K. Yan *et al.*, "Characterizing, modeling, and improving the QoE of mobile devices with low battery level," in *Proc. of MICRO*, 2015.

[11] M. R. Guthaus *et al.*, "MiBench: A free, commercially representative embedded benchmark suite," in *Workload Characterization*, 2001, pp. 3–14.

[12] U. Gupta *et al.*, "STAFF: online learning with stabilized adaptive forgetting factor and feature selection algorithm," in *proc. of DAC*, 2018.

[13] J. Chen *et al.*, "Modeling program resource demand using inherent program characteristics," in *Proc. of the ACM SIGMETRICS*, 2011.

[14] U. Gupta *et al.*, "DyPO: Dynamic pareto-optimal configuration selection for heterogeneous MpSoCs," *TECS*, p. 123, 2017.

[15] N. Mishra *et al.*, "CALOREE: Learning control for predictable latency and low energy," *Proc. of ACM SIGPLAN Notices*, 2018.

[16] Y. Zhu *et al.*, "Event-based scheduling for energy-efficient qos (eqos) in mobile web applications," in *HPCA*, 2015, pp. 137–149.

[17] T.-R. Hwang, "Battery log, version 2.0.3," https://play.google.com, 2013.

[18] T. R. Patil *et al.*, "Performance analysis of Naive Bayes and J48 classification algorithm for data classification," *IJCSA*, pp. 256–261, 2013.

[19] Hardkernel. ODROID-XU. [Online]. Available: http://www.hardkernel.com/main/main.php

[20] H. Hoffmann *et al.*, "Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments," in *Proc. of ICAC*, 2010.

[21] T. Muck *et al.*, "Adaptive-Reflective Middleware for Power and Energy Management in Many-Core Heterogeneous Systems." in *Many Core Computing: Hardware and Software, IET*, 2019.