QUAREM: Maximising QoE through Adaptive Resource Management in Mobile MPSoC Platforms

SAMUEL ISUWA, University of Southampton, UK
SOMDIP DEY, University of Essex, UK
ANDRE P. ORTEGA, Escuela Superior Politécnica del Litoral, Ecuador
AMIT KUMAR SINGH, University of Essex, UK
BASHIR M. AL-HASHIMI, Kings College London, UK
GEOFF V. MERRETT, University of Southampton, UK

Heterogeneous multi-processor system-on-chip (MPSoC) smartphones are required to offer increasing performance and user quality-of-experience (QoE), despite comparatively slow advances in battery technology. Approaches to balance instantaneous power consumption, performance and QoE have been reported, but little research has considered how to perform longer-term budgeting of resources across a complete battery discharge cycle. Approaches that have considered this are oblivious to the daily variability in the user's desired charging time-of-day (plug-in time), resulting in a failure to meet the user's battery life expectations, or else an unnecessarily over-constrained QoE. This paper proposes QUAREM, an adaptive resource management approach in mobile MPSoC platforms that maximises QoE while meeting battery life expectations. The proposed approach utilises a model that learns and then predicts the dynamics of the energy usage pattern and plug-in times. Unlike state-of-the-art approaches, we maximise the QoE through the adaptive balancing of the battery life and the quality of service (QoS) for the duration of the battery discharge. Our model achieves a good degree of accuracy with a mean absolute percentage error of 3.47 % and 2.48 % for the energy demand and plug-in times respectively. Experimental evaluation on an off-the-shelf commercial smartphone shows that QUAREM achieves the expected battery life of the user within 20–25 % energy demand variation with little or no QoE degradation.

CCS Concepts: • **Computer systems organization → Embedded systems**.

Additional Key Words and Phrases: Battery Budgeting, Maximising User Experience, Heterogeneous MPSoC, QoE-aware Resource Management, Quality of Experience, Adaptive Resource Management

ACM Reference Format:
Samuel Isuwa, Somdip Dey, Andre P. Ortega, Amit Kumar Singh, Bashir M. Al-Hashimi, and Geoff V. Merrett. 2022. QUAREM: Maximising QoE through Adaptive Resource Management in Mobile MPSoC Platforms. *ACM Trans. Embedd. Comput. Syst.* 00, 0, Article 0000 (March 2022), 29 pages. https://doi.org/xx.xxx/xxxxxxx

Authors' addresses: Samuel Isuwa, University of Southampton, Southampton, UK, si1g19@soton.ac.uk; Somdip Dey, University of Essex, Colchester, UK, somdip.dey@essex.ac.uk; Andre P. Ortega, Escuela Superior Politécnica del Litoral , Guayaquil, Ecuador, portega@espol.edu.ec; Amit Kumar Singh, University of Essex, Colchester, UK, a.k.singh@essex.ac.uk; Bashir M. Al-Hashimi, Kings College London, London, UK, bashir.al-hashimi@kcl.ac.uk; Geoff V. Merrett, University of Southampton, Southampton, UK, gvm@ecs.soton.ac.uk.

1539-9087/2022/03-ART0000 $15.00
https://doi.org/xx.xxx/xxxxxxx

1 INTRODUCTION

Smartphones have become pervasive, with over 3.5 billion users worldwide [6]. Growing reliance on smartphones and a demand for improved Quality of Experience (QoE), defined as the degree of users' perceivable satisfaction to services provided on a device, have resulted in the creation and exploitation of complex hardware platforms. These platforms typically contain heterogeneous multi-processor system-on-chips (MPSoCs) from different chip manufacturers [39]. Though various factors influence smartphone QoE [28], prior studies have found that quality of service (QoS) and battery life have the greatest impact [58]. Typically, QoS refers to objective performance measures obtained on a system or its underlying infrastructure [11, 40] without or with indirect consideration for the user of the service or infrastructure. Conversely, QoE refers to the degree of delight or annoyance experienced by a user of an application, service, or system as a result of it meeting expectations while taking into account their current state and personality [40]. While modern heterogeneous MPSoCs offer tremendous performance benefits, complex and interactive applications with diverse QoS requirements are on the increase. The advancement and ubiquity of smartphones have made them a pervasive enabler for human activity to go mobile, ranging from smart wallets and homes to connected social lives and jobs. This increases the impact on daily life (convenience) when devices are unavailable or have to be charged [37, 46]. Despite advances in chip fabrication and exponential performance improvements, energy and power management have continued to be a severe bottleneck to the seamless operation of battery operated smartphones [50, 58]. Today's sophisticated and computationally powerful smartphones rely on comparatively slow-paced developments on battery technology, for which capacity only doubles over a decade [48]. Therefore, improvements in resource management are imperative if users' QoE is to be maximised.

Considerable effort has been made by industry and academia to ensure efficient use of battery energy in smartphones. In industry, power-efficient processors that have power management (PM) such as idle state and dynamic voltage and frequency scaling (DVFS) capabilities, as well as new schedulers that are tightly coupled with DVFS governors, have evolved [48]. State-of-the-art academic research efforts have considered approaches to balance instantaneous power consumption and QoS in order to improve QoE [9, 11, 15, 22, 57, 58]. However, little research has considered how to perform longer-term budgeting of resource [17, 33, 51, 52] and, approaches that have, miss potential QoE maximisation because they are oblivious to the user's desired plug-in time (regarded as the time of the day when the user normally charges the device) and/or energy demand. This results in techniques that either fall short of users' battery life expectations (i.e., the number of hours a user uses their device before charging) or over-constrain the DVFS governor. Therefore, there is a need for resource management techniques that work with knowledge of the user's plug-in time in budgeting resources across complete battery discharge cycles and, in turn, maximise the user's QoE.

This paper proposes **QU**ality of experience-aware **A**daptive **RE**source **M**anagement (*QUAREM*), a mechanism for maximising a user's QoE of a mobile MPSoC platform during the full duration of a battery discharge (the time from plug-out to the next plug-in). It does this by identifying the QoS and energy balancing goals that maximise user satisfaction within the constraints of the expected battery life. The proposed approach consists of a model that learns and then predicts the user's energy usage pattern and plug-in times. By monitoring and predicting this at regular intervals, it enables the mechanism to perform DVFS, maximising the QoE for the duration of the battery discharge.

QUAREM makes the following contributions:

(1) A study of individual users' energy usage patterns and plug-in times while running applications, indicating considerable variation from the users.

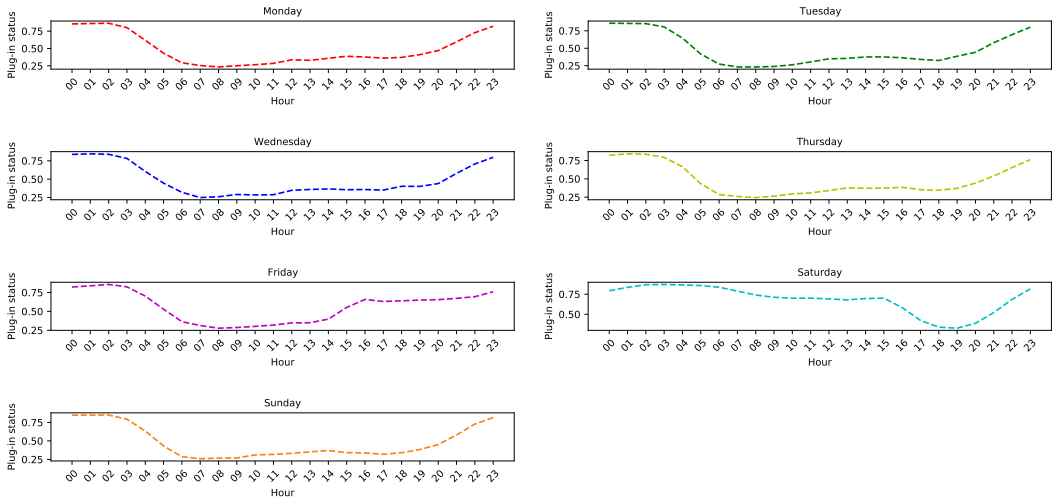

Fig. 1. Average daily plug-in status of ≈ 50 smartphone users over a period of one year. A plug-in status of 1.0 means that all 50 users had their smartphones plugged in, whereas a plug-in status of 0.0 means that all were unplugged.

(2) An exploration of energy consumption, QoS and different DVFS settings on mobile embedded devices while executing widely used real mobile applications rather than benchmarks.
(3) A model to predict the energy demand and plug-in (charging) times, with a mean absolute percentage error of 3.47 % and 2.48 % respectively.
(4) An online mechanism that monitors and performs adaptive resource management through DVFS to maximise the QoE.
(5) Evaluation on a commercial smartphone shows that our technique achieves average QoE improvement when compared with the state-of-the-art while meeting the expected battery life.

The rest of the paper is organized as follows: Section 2 presents the motivation and related works. The system architecture, prediction model and the optimisation mechanism are presented in Section 3. We present the experimental setup and the evaluation technique employed in Section 4. Section 5 presents the result with extensive experiments and show the benefit of the proposed approach in comparison to existing approaches. Finally, Section 6 concludes this paper.

2 MOTIVATION AND RELATED WORKS

2.1 Motivation

To motivate this work and to show the potential benefit of *QUAREM* in maximising users' QoE, we consider the wide range of different usage patterns and plug-in times experienced by individual users. State-of-the-art approaches have failed to incorporate these in their approach. We conduct a time series analysis on the *Sherlock dataset* [38]: real mobile data collected from ≈50 volunteer smartphone users over a three-year period. We found the dataset suitable for our analysis as it provides time-series information of host software and hardware sensors available on smartphones that can be acquired without root privileges, such as local and global application statistics and battery information. In addition, the span of years the study was conducted and its recentness (in terms of mobile device and software used) makes the data reliable compared to previous studies

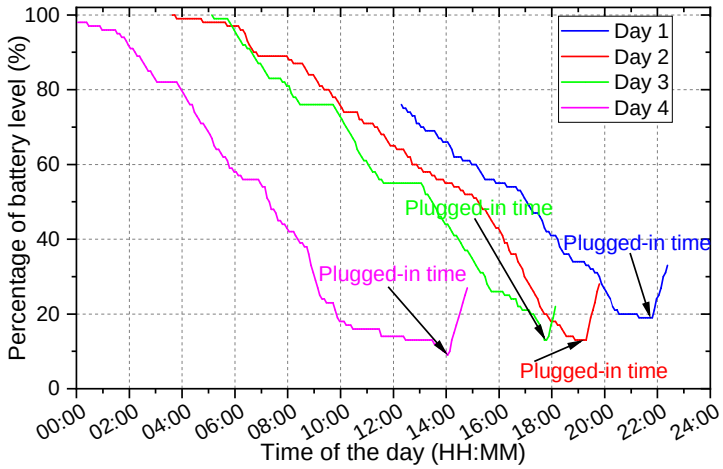

Fig. 2. Different energy usage patterns and plug-in (charging) times experience by a single user.

carried out within a short time (mostly a month or less), which is atypical to user usage and modern devices.

Fig. 1 shows the average daily plug-in status of ≈ 50 smartphone users over a period of one year as obtained from the *Sherlock* dataset. We group the Sherlock dataset per day (Monday to Sunday) per hour (00 to 23) for the whole year and then find the average of the battery plug-in status. In the dataset, all the charging options (AC, USB, wireless, etc.) are considered plugged-in (charging) and denoted with a value 1, while a value 0 denotes the device is unplugged (discharging). If more than 50 % of the users within an hour of a day, e.g. on Monday at 00 hr have an average battery plug-in status > 0.5, that hour of the day is considered plugged-in (charging). Similarly, if the average battery plug-in status of more than 50 % of the users at that time of the day over the year is < 0.5, then it is considered unplugged (discharging). In the context of the reviewed data (which is Israel), the standard work week starts from Sunday to Thursday, with Friday as a short working day and Saturday as the weekend. It can be observed from Fig. 1 that > 85 % of the users plug in their device to charge once at ≈ 22 hours with an average usage time of ≈ 15 hours on weekdays and vary on weekends as shown by the average daily plot. This indicates that users' plug-in time and usage are influenced significantly by context, such as location and time [7]. Some users exhibit erratic charging behaviour (charging inconsistently or multiple times) during the day, however, this represents < 10 % of the users and hence, not the majority. The effect of such erratic behaviour on the operation of the QUAREM approach is discussed in Section 5.4. Besides, such disruptive charging cycles can reduce the lifetime of the battery [18], thus, the need for longer battery hours management that enables both energy-saving and regular charging cycles. In addition, we also discovered that none of the users failed to utilise one of the charging options (AC, USB, etc.) within a day, i.e., users with no plug-in activity during the day do not exist except if the user's data is missing entirely for the day. It is against these backgrounds that we have decided to consider the user group who charge their devices once a day.

On further exploration of the data, we discovered wide range of different charging times and energy usage pattern exhibited by an individual. Fig. 2 shows the different energy usage pattern and charging times exhibited by a user on four different days of the week. The percentage variation reveals up to 15 % for weekdays and 25 % on weekends when compared with the average values

obtained for the week respectively. Based on the time-series descriptive analysis conducted on the *generalised Sherlock* dataset, we make the following observations:

(1) Individuals in the same user group with similar routines tend to have similar average plug-in times.
(2) Even though the average plug-in times of individuals within same user group can be similar, each user is different, and thus, tend to have unique energy usage patterns and plug-in times.

Since users' behaviour will not always be the same even for the same application use [15], we try to learn and predict these changes to provide longer-term budgeting of the constrained energy accordingly. This ensure battery life expectancy of the user is achieved within the required QoS and thus, maximising the QoE.

2.2 Related Works

Prior studies have considered the trade-off between performance and energy in mobile MPSoC platforms [16, 23, 30, 43, 44, 49]. The techniques have considered either mapping and task migration [10, 26, 31], or mapping and thread partitioning of applications [29, 45, 53] on these platforms. Similarly, imitation learning (IL) frameworks for dynamic power and performance management of applications in heterogeneous MPSoCs have also been considered [35, 36]. In [36], the authors utilise an IL-based control algorithm built at design-time from known applications to perform resource management at runtime. In [35], the authors leverage the offline policy in [36] as the starting point to bootstrap the learning of optimal policies for new applications at runtime. These approaches do not consider both battery life and user satisfaction in the resource optimisation, which are paramount to maximising QoE. Additionally, the unavailability or inaccessibility of most of the hardware performance counters and sensors utilized by these approaches makes it practically infeasible to implement on real commercial devices as they are often implemented on developmental platforms

QoE has been quantified in mobile device and exploited in trading-off energy consumption and QoS [22, 57, 59]. In Gaudette *et al.* [22], the QoE is improved by providing a satisfiable level of QoS while minimising the energy consumption using a probabilistic QoS technique. The approach in [57] focuses on mobile devices with low state of charge, which dynamically scales the cpufreq governor in a QoE-aware fashion whenever the device is at low state. Authors in [59] proposed QoS Per-Energy (QPE) metric, which quantifies the trade-off between energy and QoS and used it to explore the energy-efficient QoS for web-browser. Aside the fact that these models are built using mean opinion scores obtained from interviews, they are used for generalised QoE conditions and thus do not factor user specific data and behavioural pattern of battery usage.

To address the user specific optimisation, [58], [51] and [33] presented a model using a weighted combination of QoS and/or energy to manage system resources for each individual user. [58] provided a customised mechanism to achieve trade-off between QoS and energy toward enhancing user experience. In this approach, a novel definition of QoS based on responsiveness and display quality was first presented, then the QoE model was built (which is based on energy and QoS) and the trade-off is performed depending on the user's preference. Shamsa *et al.* [51] leveraged on the same model while incorporating individual's battery state of charge (SOC) for the trade-off. Lee *et al.* [33] exploited the non-trivial relationship between users' battery-life goal and QoE in addition to the battery status and QoS towards maximising user experience. The approach first implements a dynamic QoS scaling system that considers the battery state (termed BUQS1). Secondly, due to the inability of BUQS1 to guarantee the battery-life of user's heavy usage, a QoE model accounting for users' varying and battery- dependent QoS expectations (BUQS2) is proposed. Finally, BUQS3 improved upon BUQS1 and BUQS2 by considering learning and prediction of user behaviour for

Table 1. Summary of existing approaches compared to the proposed QUAREM approach

Reference	Perf/QoS	Power	Energy	DVFS	Battery	Plug-in time	Energy demand	Real mobile Smartphone
[24]	✓	✗	✗	✗	✗	✗	✗	✓
[25]	✗	✓	✗	✓	✗	✗	✗	✓
[4]	✓	✗	✓	✓	✗	✗	✗	✓
[43]	✓	✓	✓	✓	✗	✗	✗	✗
[9]	✓	✓	✓	✓	✗	✗	✗	✓
[33]	✓	✓	✓	✓	✓	✗	✓	✗
[51], [58]	✓	✓	✓	✓	✓	✗	✗	✗
[52]	✓	✓	✓	✓	✓	✓	✗	✗
QUAREM	✓	✓	✓	✓	✓	✓	✓	✓

the purpose of continuous rebalancing of the energy within the mobile device. These approaches do not consider individual user plug-in times thus, missing potential opportunity to maximise the QoE by either falling short of the battery life goal or over constraining the cpufreq governor. In addition, a recent study [7] and our analysis have shown that there is significant variation in users' usage pattern and mostly dependent on context (such as time and location) as against the fixed hours and SOC levels utilised in BUQS [33] and [51] respectively.

Recent work [52] has considered user preference and user plug-in/out pattern in maximising QoE and battery cycle life (BCL) by minimizing the energy consumption when the state of charge (SOC) is low and plug-in event is not soon. The approach does not consider minimising the total energy to meet the user battery life expectancy in all situations thus, making the approach agnostic to energy demand or budget for the predicted plug-in time. In this paper, we maximise the QoE by adaptively allocating resources at runtime taking into consideration the plug-in time and the energy demand of the user based on the historic and current usage pattern. Table 1 shows the summary of the existing works in comparison to the different metrics considered. The symbol ✓against a metric shows that the approach considered the metric while the symbol ✗ denotes that the metric is not considered in the approach. As shown in Table 1, QUAREM approach considers all the metrics in the table while other approaches do not consider all and/or are implemented on generational developmental platforms rather than state-of-the-art devices.

Several QoE models based on users' mean opinion scores (MOS) with the QoS measure have been presented due to the diversity of QoS perspectives and the need to consider the user's perspective. Popular among the proposed models is the exponential **I**nterdependency of **Q**uality-of-e**X**perience and quality-of-service (IQX) [19], which works well with accurate insight to interruptions in system-level QoS. In the model, QoE is divided into three distinct regions using X_1, and X_2 (optimum and minimum points, respectively). The regions include constant optimal QoE ($QoS < X_1$) where QoE is maximum; acceptable QoE ($X_1 \leq QoS < X_2$) where the QoE can be tolerated by the user, and the unacceptable region ($QoS \geq X_2$) where the QoS becomes unbearable and the service is given-up by the user. More recent QoE models or improvements have evolved either considering aspects fundamental to the user (e.g., battery life) or the user's characteristics and context influencing factors. For example, Lee *et al.* [33] improved the IQX model [19] to incorporate the battery state and award a penalty to show user dissatisfaction as a result of total battery discharge with web-page loading time as the QoS metric. They also exploited different X_2 values based on the users' varying and battery dependent QoS expectations, since the model allows X_2 to be varied relatively large

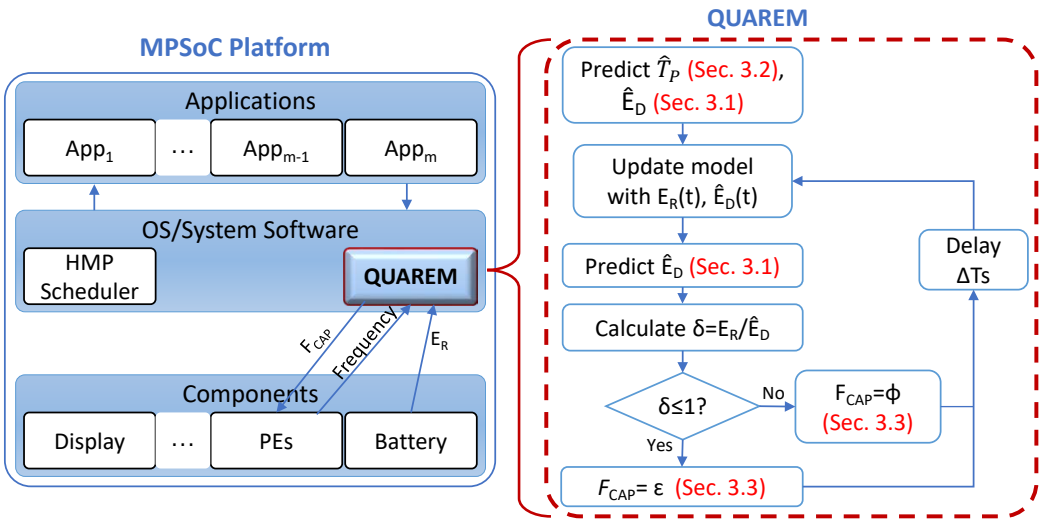

Fig. 3. Proposed system overview of the QoE-aware adaptive resource management methodology showing its position on the MPSoC platform (left) and the adaptive mechanism (right).

compared to the other model parameters. We leverage this improved IQX model and applied a penalty of 0 rather than -100, to show users' dissatisfaction when battery life expectation is missed. However, rather than considering web-page loading time, we utilise *frame rate janks* [8, 48] as our QoS measure, an important metric for interactive systems, which refers to the number of frame deadlines missed (dropped frames), since most mobile applications are interactive in nature. The model is thus expressed as follows:

$$QoE = \begin{cases} QoE_{max} & \text{if } QoS < X_1. \\ \alpha \cdot e^{-\beta \cdot QoS} + \gamma & \text{if } X_1 \leq QoS < X_2. \\ 0 & \text{if } X_2 \leq QoS \text{ or if battery is fully depleted.} \end{cases} \quad (1)$$

where α, β, and γ are 85.96, -0.347, and 27.8, respectively [33]. Similarly, QoE_{max}, and X_1, are defined as 100, and 0.5, respectively. However, for the X_2, we utilised different values depending on the ratio of the remaining energy to the energy demand (Eq. 11), as will be explained in Section 3.3, which determines the required settings.

3 THE QUAREM METHODOLOGY

This section presents the proposed **QU**ality of experience-aware **A**daptive **RE**source **M**anagement (*QUAREM*) methodology. Fig. 3 shows the proposed *QUAREM* methodology. It consists of its position within the MPSoC Platform (Fig. 3 (left)), and the techniques involved (Fig. 3 (right)). To achieve our desired objective of maximising QoE through energy budgeting and QoS balancing, we must be able to predict the energy demand until the next plug-in time. The overall implementation is dependent on the users' energy usage behaviour, i.e. plug-in time and energy demand, which is dynamic. These are the novel features of QUAREM:

(1) Prediction of energy demand until the next plug-in time, $\hat{E}_D$, to deliver the QoE required based on user's usage. This estimates the overall energy requirement for the user's expected battery life before plug-in (Section 3.1).

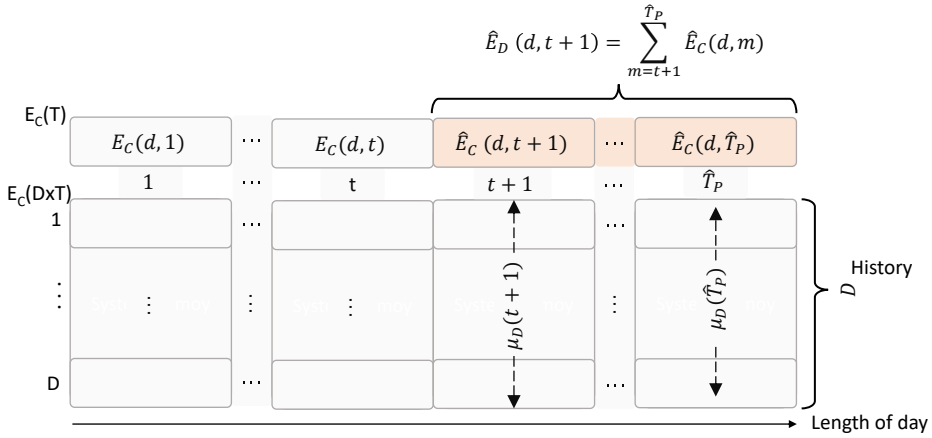

Fig. 4. Pictorial representation of the energy demand $\hat{E}_D$ prediction (Adapted from [1, 32]).

(2) Estimation of the time of the next plug-in, $\hat{T}_P$: This is the expected time-of-day the user normally plugs-in the device to charge it, e.g. 19:12 (Section 3.2).
(3) Determination of the remaining energy, E_R in the battery.
(4) Monitoring and adaptive effective configuration of DVFS for the CPU cores (Section 3.3).

To adaptively manage the resources, the above information ($\hat{T}_P$, $\hat{E}_D$ and E_R) allows the necessary DVFS settings, F_{CAP}, to be configured to ensure the desired QoS while meeting the expected user's battery life. F_{CAP} is determined based on the ratio of E_R and $\hat{E}_D$, referred to as δ, and the correlated frequency parameters given as ϵ and ϕ, which determine the DVFS settings for the different energy requirements and the optimum DVFS setting respectively (discussed further in Section 3.3). The whole process is repeated after a delay interval of ΔT_S, which is obtained by performing extensive experiments and leading to efficient results.

3.1 Energy Demand Prediction

A fast convergence and most reliable algorithm for building a time-series forecasting model based on a set of data is exponential smoothing [1, 27]. Based on its popularity, quick convergence, adaptiveness and considerable level of accuracy at low overheads in terms of memory and computation, the exponential weighted moving average (EWMA) is considered for the energy demand prediction. By using this approach, the model gives a reasonable forecasting result with fast convergence and more implication to the latest data. It also solves memory overhead encountered by most methods through its recursive technique of constantly updating value in memory with the recent observation. The following describes the implementation of the model.

3.1.1 Exponential Weighted Moving Average (EWMA). The user model is built to predict the energy demand of an individual user until plug-in $\hat{T}_P$, based on the user's energy usage pattern and behaviour using EWMA. The EWMA model is trained using the user's historical data collected and updated online to ensure accurate prediction.

To effectively predict the energy demand, $\hat{E}_D$, the time of the day is divided into equal number of discrete time steps of length ΔT_S, such that the vector $\{E_C(1), E_C(2), \ldots, E_C(t), E_C(t+1), \ldots, E_C(\hat{T}_P)\}$ represents the energy consumed at each time step of smartphone operation for the current day while the matrix $E_C(D \times T)$ represents the energy consumed in the discrete time steps for the last $D \in \mathbb{Z}^+$ days (as shown in Fig. 4). These equal discrete time steps (for instance, 24 h

can be divided into 288 discrete time steps with $\Delta T_S = 5\,\text{min}$) are considered so that the model can efficiently record and refer to the historical E_C data by consistent indices. These E_C values are computed using Eq. (2).

$$E_C = E(SOD_{V2} - SOD_{V1}) \tag{2}$$

where E is the rated battery energy capacity for the smartphone and SOD_{Vi} is the state-of-discharge of the battery at V_i (i=1 or 2), i.e. in current and previous time steps, which can be computed as the ratio of the amount of charge removed from the battery at the given state, Q_d, to the total amount of charge, which can be stored in the battery, C, (usually expressed as a percentage $\frac{Q_d}{C} \times 100$) [55]. These values (battery level, battery capacity, battery status, charge, voltage etc.) are accessible from the battery service state, which provides estimate of these values. These E_C values are recorded at every time step and then used to predict the required energy in the subsequent time steps, $\hat{E}_C$. For high prediction accuracy, typically many samples back in time are considered, e.g. for predicting energy for $t+1$ at the current time t, samples at t, $t-1$, $t-2$, etc. are used. To illustrate this, in determining the energy demand, $\hat{E}_D$, for time step $t+1$, at the current time step t and day, it involves the cumulative addition of the predicted energy consumption $\hat{E}_C$, as shown by the coloured area of Fig. 4 and given by Eq. (3).

$$\hat{E}_D(d, t+1) = \sum_{m=t+1}^{\hat{T}_P} \hat{E}_C(d, m) \tag{3}$$

where $\hat{E}_D(d, t+1)$ is the predicted energy demand at time $t+1$ and $\hat{E}_C(d, m)$ is the predicted energy consumption at time step m, which is predicted using the EWMA. The EWMA prediction of the future energy consumption, $\hat{E}_C$, at time step $t+1$, is computed by taking the actual energy consumed, $E_C(t)$ at the current time step t with weighting α and updating it with the previous averaged energy consumptions at time step $t+1$ for the previous D days considered with weighting $1-\alpha$. Thus, $\hat{E}_C(d, t+1)$ is given by the first part of Eq. (4) where $m = t+1$. Alternatively, in the following time steps when $t+1 < m \leq \hat{T}_P$, that is $\hat{E}_C(d, t{+}2), \hat{E}_C(d, t{+}3), \ldots, \hat{E}_C(d, \hat{T}_P)$; $E_C(d, m{-}1)$, is replaced with the estimated $\hat{E}_C(d, m-1)$ where it becomes the second part of Eq. (4).

$$\hat{E}_C(d, m) = \begin{cases} \alpha E_C(d, m-1) + (1-\alpha)\mu_D(d, m), & m = t+1. \\ \alpha \hat{E}_C(d, m-1) + (1-\alpha)\mu_D(d, m), & t+1 < m \leq \hat{T}_P. \end{cases} \tag{4}$$

Unifying Eq. (4) for general situations, we have Eq. (5)

$$\hat{E}_C(d, m) = \sum_{k=t+1}^{m} \left\{ (1-\alpha)\alpha^{m-k} \mu_D(d, k) \right\} + \alpha^{m-t} E_C(d, t) \tag{5}$$

where α is the smoothing constant determining how the historic and current data are weighted, with value $0 \leq \alpha \leq 1$. $\mu_D(d, k)$ is the mean energy consumed at time step k for the previous D days, which is given by Eq. (6).

$$\mu_D(d, k) = \frac{1}{D} \sum_{m=d-1}^{d-D} E_C(m, k) \tag{6}$$

Therefore, substituting Eq. (5) into Eq. (3) and solving for μ_D from Eq. (6), we have Eq. (7) for the energy demand prediction.

$$\hat{E}_D(d, t+1) = \sum_{m=t+1}^{\hat{T}_P} \sum_{k=t+1}^{m} (1-\alpha)\alpha^{m-k} \left(\frac{1}{D} \sum_{m=d-1}^{d-D} E_C(m,k) \right) + \alpha^{m-t} E_C(d,t) \tag{7}$$

Appropriate values of α (weights), number of time steps for the day and D (previous days) are selected such that the prediction error is minimised while considering the constrained device and the overall goal of maximising the user QoE across the battery discharge. We first considered the average operating hours of users based on our analysis from Section 2 to determine an optimum duration for the time step (or length for the time steps, ΔT_S) and D days, in which the constant monitoring and adjustment of the QUAREM approach would not be with an additional energy and memory cost. A selective search strategy was used for the parameters that best optimise our goal. For example, [10 s, 60 s, 300 s, 1800 s, 3600 s] are considered for ΔT_S while $D = [5, 10, 20, 30, 40]$ days were also considered. Then an investigation of varying α values $[0, 1]$ was also carried out while considering the prediction errors.

Our proposed model accuracy is evaluated using Mean Absolute Error (MAE). We also used Mean Absolute Percentage Error (MAPE), which estimate the error in percentage of the actual value for better representation of the level of accuracy. The MAE and MAPE are given by Eq. (8) and (9), respectively.

$$MAE = \frac{1}{T} \sum_{m=1}^{T} |E_D(m) - \hat{E}_D(m)| \tag{8}$$

$$MAPE = \frac{1}{T} \sum_{m=1}^{T} \left| \frac{E_D(m) - \hat{E}_D(m)}{E_D(m)} \right| \tag{9}$$

where $\hat{E}_D$ is the predicted energy demand and E_D is the actual energy demand by the smartphone while operating. A time delay ΔT_S of 5 min, $D = 4$ *days* and $\alpha = 0.3$ tend to be the optimised values for the parameters that gives minimal error. The energy demand prediction algorithm is not an offline process but online too - updated over a granularity of ΔT_S (5 min) based on the additional information gained from the user's behaviour within that time step until the plug-in time.

3.2 Plug-in Estimation

Considering our analysis of the *Sherlock* and individual data (Section 2.1), we assume that the user has a similar average plug-in status; however, with a variation of up to 25% thus, we use EWMA to estimate the user's plug-in time (charging) for a day. Though a simple moving average can be used to estimate the plug-in times, it does not respond to the user's changes within different days quickly. Thus EWMA is more suitable. The historic data collected from the user's plug-in behaviour shows that this plug-in time can be estimated since the user normally uses the smartphone during the day and plugs-in (charges) during the night when there is adequate availability of tethered power. Thus, the next plug-in time can be estimated using the user's historic data. The next plug-in time, $\hat{T}_P(t+1)$, is estimated using EWMA as shown in Eq. 10.

$$\hat{T}_P(t+1) = \alpha T_P(t) + (1-\alpha)\hat{T}_P(t) \tag{10}$$

where α is the smoothing constant where $0 \leq \alpha \leq 1$. The higher the value of α, the lower the smoothing. It is worth mentioning that the plug-in prediction algorithm is not entirely an offline process and not online over the granularity of a per hour process, but every day it is recalculated based on that additional information gained from the user behaviour that day. So, if, for instance, you always use your phone and find a plug based on your SOC level at 2 pm, QUAREM starts to

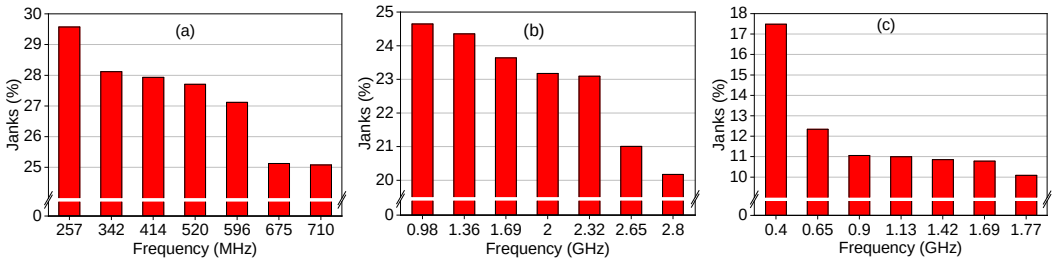

Fig. 5. Janks (dropped frames) observed for Google Maps while varying the frequency of the (a) GPU (b) big CPU and the (c) Little CPU cores.

learn that you plug-in at that time and therefore, overtime your plug-in time does take the SOC because of your behaviour.

3.3 Adaptive DVFS Setting

From our exploration, we discovered that the 'Schedutil' governor, which integrates the DVFS mechanism within the scheduler, is widely used in most recent smartphones due to its compatibility with modern heterogeneous multi-processing (HMP) schedulers [13, 21]. Studies have shown that the processor consumes most of the Smartphone's energy [12, 42]. For instance, Chen *et al.* [12] have shown that processors consume around 40% of the energy while playing mobile games on smartphones. In [42], the authors have demonstrated that >50% of the energy is consumed by the processor at reduced screen brightnesses. Similarly, a recent study on the power consumption of smartphone battery and energy usage also reveals that the CPU is among the power-hungry components in smartphones [46]. Thus, if DVFS is utilised properly, it has the potential to save energy with little or no QoS degradation. For example, Figs. 5 and 6 show the occurrence of *frame rate janks* (janks), which is the frame deadlines missed, while changing and capping the frequency on a mobile MPSoC platform. We used the 60 frame per second (FPS) for computing the frame rate janks, which means all processing (computing, networking, and rendering) must be completed within 16.7 ms per frame to avoid the occurrence of a jank. The average jank value was considered for the entire period of a productive task for each application e.g. Google Maps, YouTube, (discussed in Section 4.1 and Table 2) across the several runs. In Fig. 5, the Google Maps workload is used while varying the frequency of the GPU at fixed lowest frequencies for the big and LITTLE CPU (Fig. 5a); varying the frequency of the big CPU at fixed lowest frequencies for the LITTLE CPU and GPU (Fig. 5b); and varying the frequency of the LITTLE CPU at fixed lowest frequencies for the big and GPU cores (Fig. 5c), respectively. From the three figures, while using other DVFS governors, a significant increase in frame rate janks can be observed as the frequency is lowered for both the CPU and the GPU. This is because of the lack of synergy between the DVFS governor and the scheduling mechanism [21], thus, causing slow frequency ramping and task migration to the right cores.

Fig. 6 shows the janks observed for YouTube, Google Maps and Chrome while capping the frequency of the big CPU cores. This is not showing any new management approach, but it shows the impact of capping the frequency while using *Schedutil* governor. The result shows a lower jank percentage even at lower frequency compared to fixing the frequencies in Fig. 5. This further shows the advantages of schedulers that are tightly coupled with the DVFS governor as they have awareness of the individual task's load in making better scheduling decisions. Thus, the QUAREM

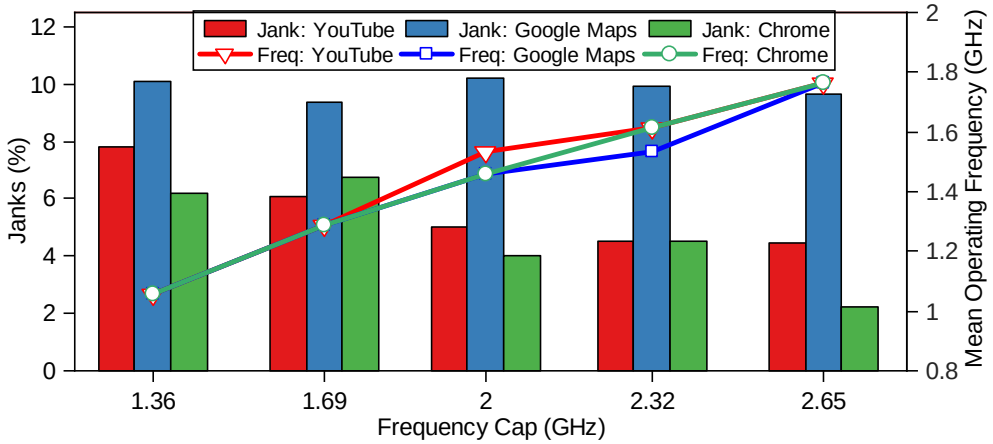

Fig. 6. Janks (%) and mean operating frequencies (GHz) observed for YouTube, Google Maps and Chrome workloads while capping the frequency of the CPU.

approach uses the state-of-the-art DVFS governor - *Schedutil* already employed in the current smartphone generation.

Therefore, in meeting our objective of maximising the QoE while ensuring the battery life expectation within the required QoS, we propose an adaptive technique utilising DVFS which works as shown in Fig. 3 (right). First, an offline exploration on the commercial smartphone was carried out. Various workloads were executed using the workload automation setup with ≈ 150 runs per workload to determine the energy consumption and the frame rate janks (QoS) at different frequency cap for the big CPU cores as represented in Fig. 6 and Fig. 7. The foreground applications are executed one at a time, however, a range of background services are also running in the background, as is the common case with smartphones operation. This scenario is considered because the uncontrolled energy consumption by background applications in the past has been aggressively addressed in Android 8 and above [14]. From the exploration, we were able to determine the optimum capping frequency ϕ for the big CPU cluster, as it is not beneficial to operate the device at maximum frequency due to power and thermal concern, which is in agreement with the findings of Li *et al.* [34]. In finding the optimum capping frequency, ϕ, (the minimum frequency where QoE remains maximum), we leverage the improved IQX model employed, which divides the QoE into three distinct regions using X_1, and X_2 (optimum and minimum points respectively). The regions include constant optimal QoE where $janks < X_1$, acceptable QoE region, which is also known as the sinking QoE where janks is given by $[X_1, X_2)$, and the unaccepted region where $janks \geq X_2$. Since users are already satisfied with the QoS level at the constant optimal QoE region, the provision of additional QoS by way of increasing the frequency does not necessarily lead to higher QoE. Therefore, the optimum capping frequency ϕ is determined at this point. We also determine the different frequency caps depending on the energy requirement while considering the QoS, which we denote as ϵ. For example, in our particular setup, which involves a Google pixel 3 smartphone, we determined the value of ϕ as 2.65 GHz while the determined ϵ values are 2.40 GHz, 2.00 GHz and 1.69 GHz for $0.85 \leq \delta \leq 1$, $0.6 < \delta < 0.85$ and $\delta \leq 0.6$, respectively.

At runtime, the *QUAREM* approach consistently monitors, updates the model at each time delay ΔT_S of the time step T and allocate the resources accordingly to achieve the overall objective. For instance, when the energy in the battery is predicted to be insufficient to sustain the operation

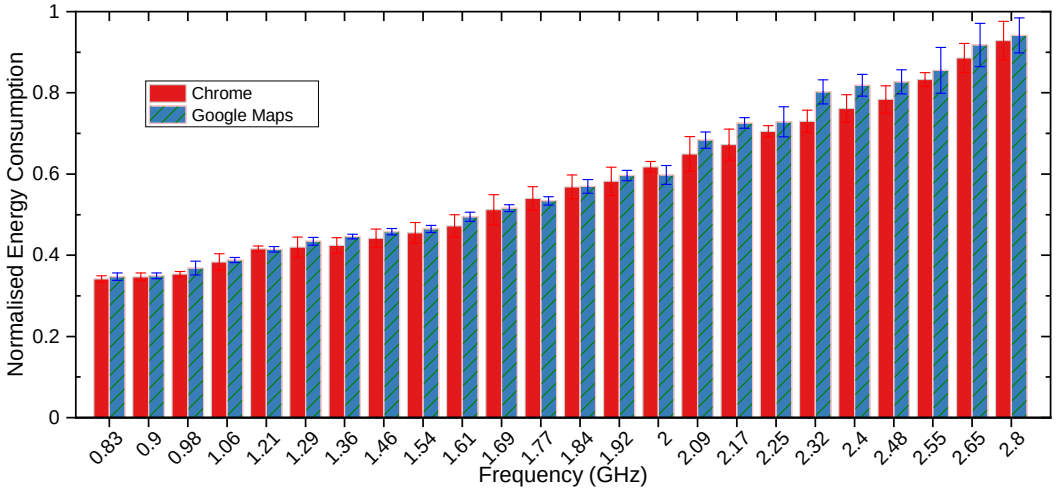

Fig. 7. Normalised energy consumption for Google Maps and Chrome while capping the frequency of the big CPU cores.

of the smartphone until $\hat{T}_P$, our system monitors and reconfigures the DVFS settings by capping the frequency, F_{CAP} of the CPU cores at each time-step (represented as delay ΔT_S in Fig. 3). The F_{CAP} value is determined by $\delta(t)$, which is the ratio between the remaining energy and the energy demand at time step *t+1* and is given by Eq. (11).

$$\delta(t) = \frac{E_R(t)}{\hat{E}_D(t+1)} \tag{11}$$

where E_R is the estimated remaining battery energy in the smartphone at time step t, which is given as the product of the battery level, B_L, and the battery capacity, B_C, (i.e. $E_R(t) = B_L(t) \times B_C$) and $\hat{E}_D$ is the predicted energy demand. Since users' behaviour and SOC level will not always be the same even for the same application use, QUAREM also considers the level of the battery SOC at every time step of the device operation until plug-in in addition to the energy consumed and predicted energy demand before taking decisions. The idea of updating the model at every time step ensures that these dynamic changes are learned and addressed online to provide longer-term budgeting of the constrained energy within the required QoS and thus, maximising the QoE. However, if at much lower remaining battery than the predicted energy demand and the user feels s/he has charging options or pressing task with a need for a higher frequency, there is an option to disable QUAREM so that the application can operate uncapped. When the user turns it back on afterwards, QUAREM will attempt to manage the remaining battery. In essence, $F_{CAP} = \phi$ if the value of $\delta(t) > 1$, implying that $E_R(t) > \hat{E}_D(t+1)$. Similarly, F_{CAP} will be the different sets of frequencies if the value of $\delta(t) \leq 1$, implying $E_R(t) < \hat{E}_D(t+1)$. Thus, F_{CAP} is expressed as Eq. (12).

$$F_{CAP} = \begin{cases} \phi & \text{if } \delta(t) > 1. \\ \epsilon & \text{if } \delta(t) \leq \mathbf{1}. \end{cases} \tag{12}$$

where ϕ is the optimum capped frequency of the MPSoC as determined from our offline exploration of the different applications while capped at different frequencies; and ϵ (implying $E_R(t) < \hat{E}_D$) is the calculated values based on the relationship between the energy consumption

and the frequency of the CPU cores as shown in Fig. 7. Note that the F_{CAP} setting is implemented per cluster since the smartphone utilised provides only per cluster DVFS control (as is typical in mobile SoCs). The firmware automatically adjusts the voltage based on preset pairs of voltage-frequency values with every variation in frequency to achieve the required energy saving. Since the firmware handles the frequency switching, the switching overhead would not be more than usual. Moreso, the experimental results measure the actual energy consumption, and therefore any overheads are inherently included. Details about the mobile hardware and the operating frequencies are given in Section 4.1.

4 EXPERIMENTAL SETUP AND EVALUATION

This section presents the experimental setup which consists of the mobile hardware platform, tools and the workloads employed for the experiment and exploration. It also discusses the relevant existing state-of-the-art approaches used for comparison.

4.1 Experimental Setup

4.1.1 Mobile Hardware. The experiment was conducted on a Google Pixel 3, a flagship commercial smartphone and third-generation design by Google that contains the Qualcomm Snapdragon 845 SoC (SD845) [20, 54]. We used the Google Pixel 3 that is two years old because of the predominance of the chipset category and its advancement over many smartphones in the market. The mobile chipset technology used, which falls within the mid-tier category of smartphones, represents the highest category in the market compared to low- and high-end categories [41], thus making it an up-to-date technology in the market. For instance, out of the estimated 1.6 billion smartphones projected to be sold by 2021, ≈ 580 million (36.3%) are mid-tier smartphones, surpassing high and low-end ones that are 530 million (33.1%) and 490 million (30.6%), respectively. Real mobile smartphone was used as opposed to development board to address the fundamental challenge associated with the lack of scalability of board-based implementations to real mobile devices either due to unavailability of sensors used or impossibility of accessing them on actual smartphones. In addition, QoE is subjective, and most of the factors in real mobile operation are not captured (such as the interactivity of the user and other IP cores within the SoC) due to the fact that the experiment is often carried out in a controlled environment and/or simulated based on some assumptions that are typically not true representations in real-life situations [48]. The SD845 integrates performance (big) and efficiency (LITTLE) CPU clusters alongside a Qualcomm Andreno 630 GPU cluster. The big cluster consists of 4 × Kryo 385 gold CPU cores that allow cluster-wise DVFS with 24 operating points ranging from 825–2803 MHz. Similarly, the LITTLE cluster consists of 4 × Kryo 385 silver cores with 18 operating points ranging from 300–1766 MHz [47, 54]. The Andreno 630 GPU cluster also support cluster wise DVFS with seven frequency values: 257 MHz, 342 MHz, 414 MHz, 520 MHz, 596 MHz, 675 MHz and 710 MHz.

4.1.2 Software Infrastructure. The Google Pixel 3 runs the Android 11 operating system. The scheduler, based on Qualcomm's custom HMP scheduling, is tightly coupled with the DVFS governor (*Schedutil*) [20]. In addition, it supports DVFS by editing relevant files in the sysfs directory from the userspace in rooted devices. The availability of these resources and next generation software makes this platform ideal for our experiment.

To facilitate the experiment on this device, *Workload Automation* (WA) was used, which is a developer framework that helps execution of mobile application and collection of measurements on mobile embedded devices running Android and Linux in a repeatable manner [3]. The flexibility and extensibility of this tool makes it ideal for exploring an off-the-shelf mobile platform to better

Table 2. Summary of the applications used in the experiment and the productivity task performed by each based on workload automation [3]

Application	Class	Productivity Task
Chrome	Web based task	Search and navigate through couple of web pages.
Gmail	Email	Creating and sending emails with task such as opening the app, clicking on the create email field, searching for the file to be attached and entering the details in the required fields (recipient details, subject and compose field), and then clicking on the send button.
Google Maps	Navigation task	Search for locations, pan and zoom around the map while following the path along the route.
YouTube	Video	Search and play video within the YouTube application. While playing the video, tasks such as video seeking, pausing playback and navigating to the comments section are done.

understand the interactivity between the user, applications and the SoC in a way that it can easily be repeated and measured.

4.1.3 Workloads. Popular mobile applications supported by *WA* such as Chrome (browser), Google Maps, Gmail and YouTube were selected and executed. These applications are among the top 10 most widely used in smartphones because of their relevance in users' daily lives [5], - also considered among the top 10 apps people cannot live without in the United States. They also load and operate differently from each other, thus exhibiting a wide range of characteristics. For example, while the YouTube application streams data to the application for the video to be rendered, the Chrome application involves a client-server exchange of data based on the user's action/request. Due to this and the difference in their QoS requirement, they were selected for the experiment. Similarly, the use of real mobile applications which are multi-platform based enables their applicability across many possible hardware configurations of a heterogeneous multi-core chip-set. Table 2 summarises the applications used, their classes and specific productivity task.

4.2 Evaluation

4.2.1 Comparison. The proposed approach was compared with the following existing state-of-the-art approaches to demonstrate its QoE maximisation while meeting the battery life expectation of the user.

(1) Schedutil with Energy-aware scheduling (DF) [4]: Recent HMP are equipped with EAS, which integrate the cpufreq governor (DVFS) in the scheduling process. This approach focuses on optimising energy consumption by leveraging on the different power and performance capabilities of the multi-cores within the SoC. This is considered for comparison because of its popularity in flagship commercial smartphones as it provides energy efficiency and user satisfaction by balancing the trade-off between energy and QoS.
(2) Powersave Governor (PS) [25]: This approach focuses on minimizing the power consumption by ensuring the power hungry cores are operated at low frequency. Although the implementation of Powersave on the experimental platform occasionally allows the CPU frequency to scale to the maximum frequency (2.80 GHz) depending on the workload requirements [56], it predominantly operates at low frequency (825 MHz) as shown in Fig. 8. This is selected as a comparison candidate to show the energy saving of our approach with respect to it.

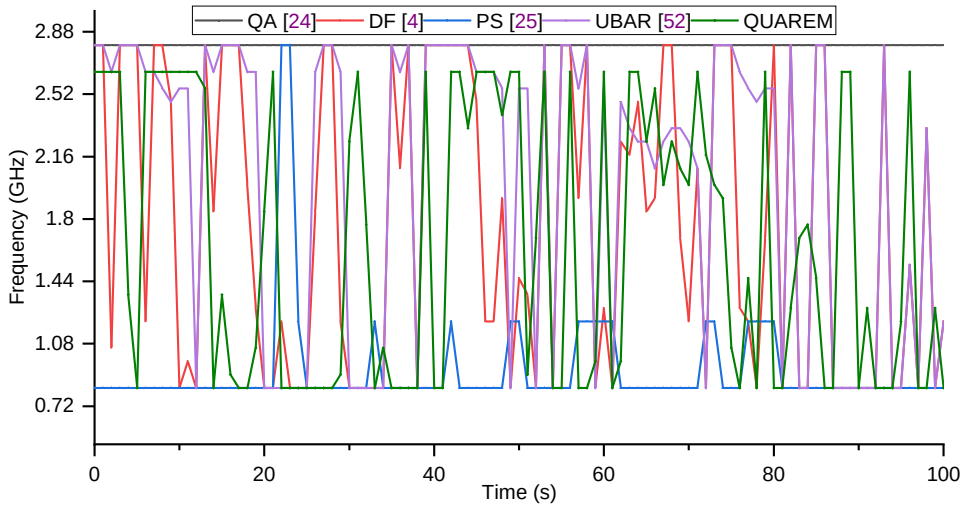

Fig. 8. CPU frequency traces of the different approaches while running Google Maps application.

(3) QoS-Aware (QA) [24]: This approach is aimed towards maximising the QoS by assigning high frequency to the cores. This is readily available on the commercial smartphones and operate in similar way to the Linux performance governor.
(4) User- and Battery-aware Resource management (UBAR) [52]: This approach focuses on maximising both QoE and aging by considering users' preferences, plug-in/out patterns and state of charge (SOC) level. The resource allocation/management for each user is based on the predicted user's action (plugged-in) and estimated current SOC from the user and SOC model. This state-of-the-art work is the most relevant to our work. The energy minimization strategy used in this method is reduction of performance reference (or performance target). First, when SOC is high (above 70%), applications performance requirement are set to the highest reference. Secondly, when SOC is less than 70%, performance target is relaxed gradually, 80% in the first instance and to 50% subsequently.

4.2.2 Usage Pattern. To experimentally evaluate the proposed approach at runtime, use-case scenario as observed by the usage pattern in one of the days was considered. Workload automation (WA) which allows interactive execution of mobile application on real mobile device in a way that it can easily be repeated and measured as discussed in Section 4.1.2 was utilised. The usage sequential pattern considered for the evaluation while running the mobile workloads on the mobile hardware using the WA is given in Table 3. In the table, the first column shows the action or operation carried out by the user, whereas the second column shows the application executed or the time taken for the action. For instance, row 3 shows that the smartphone is idle (not used by the individual) for about 25 minutes. Similarly, the email action involves all the tasks in sending an email, ranging from the launching of the app (Gmail) to clicking on the send button.

The use-case of each of the application considered in the experiment is in conformity with the productivity task of the WA as described in Table 2. To further illustrate the adaptability of the different approaches to changes in either the user usage pattern or the energy requirement, we considered the scenario in Table 3 at different initial battery levels ranging from 100–40 % (varying from 0-60) represented as 100%, 90%, 80%, 70%, 60%, 50% and 40%. The rationale behind varying the initial SOC level is to show how the QUAREM resource management adjusts to different scenarios

Table 3. The usage sequential pattern considered for the evaluation while running the mobile workloads on the mobile platform using the workload automation

Action	Application/Time
Email	Gmail
Check news online	Chrome <22 minutes>
Idle	25 minutes
Check information online	Chrome
Backlit on while reading the content	20 minutes
Set Google Maps and drive to work	Google Maps <69 minutes>
Idle while occasionally checking the screen	185 minutes
Watch video	YouTube <20 minutes>
Drive back home	Google Maps <101 minutes>

in real-life as users might not always begin the day with 100 % battery level and also to show how the approach will adapt to varying energy requirements.

5 RESULTS AND DISCUSSION

5.1 Prediction Results

To facilitate the learning and prediction of the energy demand until plug-in time, the model uses individuals' user data on energy usage patterns. This was collated using *Funf Journal* [2], an Android application installed on a smartphone to record the energy usage pattern over three weeks. To avoid memory overheads and battery consumption due to many different sensor samples at a relatively high frequency, we only enabled the battery sensor every 5 min since we are interested in the energy usage and plug-in time of the individual over a long time period.

The data collected consists of the Unix timestamp, battery level, voltage, plug-in status, charging type and battery status. We split the user's dataset into training (two weeks of data) and testing set (one week). Two weeks worth of collected data are used to train and fit our model as detailed in Sections 3.1 and 3.2. The extra one week of data is used to evaluate (test) the experimental results. While user behaviour may vary in practise, depending on the DVFS/scheduling governor, for the purposes of this study we assumed that it does not and thus varied the initial battery level to account for those changes.

Fig. 9 shows the predicted plug-in time of our model compared to the actual plug-in time of the user. The prediction is done at the moment when the user unplugs the device from charging (which varies per individual). The predicted times show good level of accuracy with error ranging from 0–12 min for over half of the days. The results give a mean absolute percentage error of 2.48 %, which is equivalent to 22.3 min. This error is small compared to the total hours in a day ($\approx$ 15 h) the user spent before the next plug-in time, as shown in Fig. 1.

Fig. 10 shows the predicted energy demand, $\hat{E}_D$, and the actual energy demand, E_D, of the user within a day. This value is updated every 5 min (each time step) with the most recent value of the energy consumption (E_C) until the predicted plug-in time ($\hat{T}_P$) as enumerated in Section 3.1. From the results, we can see that our model does well in predicting the energy demand within the day with a mean absolute percentage error of 3.47 %.

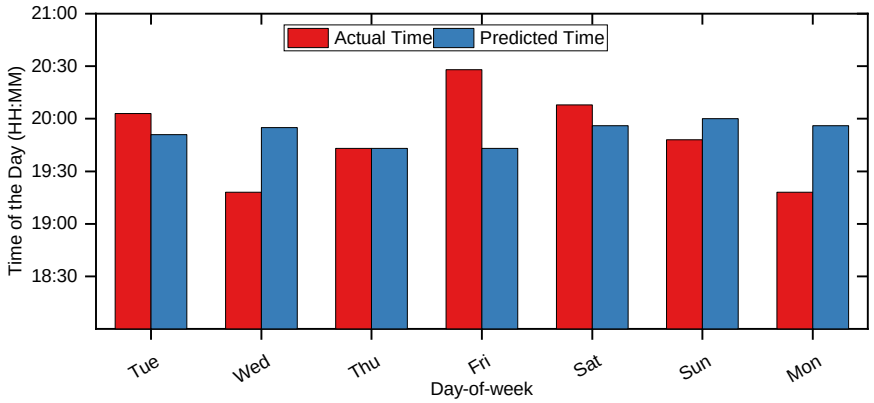

Fig. 9. Comparison between *Predicted plug-in time* vs the *Actual plug-in time* of the user.

5.2 Battery Life Extension

Fig. 11 shows the battery discharge profile for the five different approaches considered for the scenario described in Table 3. Similar results are expected in other scenarios as in Section 4.2.2 because the approach has considered varying workloads and users of varying nature. The figure shows the different percentages of the battery level (the remaining energy) for the five approaches at the end of the execution while the user starts the day with 100% initial battery charge. It can be seen that all the approaches are able to meet the expected battery life of the user with 7%, 18%, 30%, 24% and 22% final battery level still remaining for the QA, DF, PS, UBAR and QUAREM techniques respectively. As we further decrease the battery's state of charge at the start of the day as reflected in Fig. 12, QA, DF and UBAR approaches fail to meet the battery life at 90%, 80% and 80% state of charge respectively. QA failed at 90% initial battery level because of its unnecessary allocation of higher frequency to performance cores in the quest to deliver maximum QoS as shown in Fig. 8. Though DF approach maps resources to energy-efficient cores, it still fails at 80% initial battery level due to its unawareness of the user's battery life goal. Similarly, UBAR slightly fails to meet

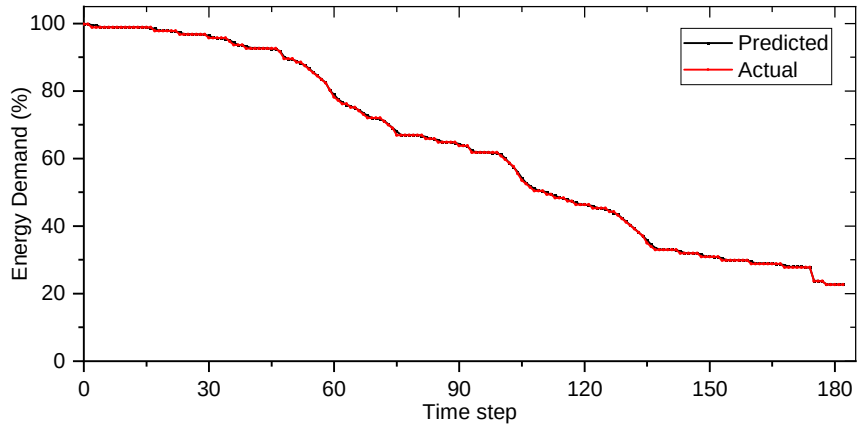

Fig. 10. Comparison between the *Predicted energy demand* $\hat{E}_D$ vs the *Actual energy demand* of the user across the time steps for a particular day. The energy demand plot is normalised as a percentage of the battery level.

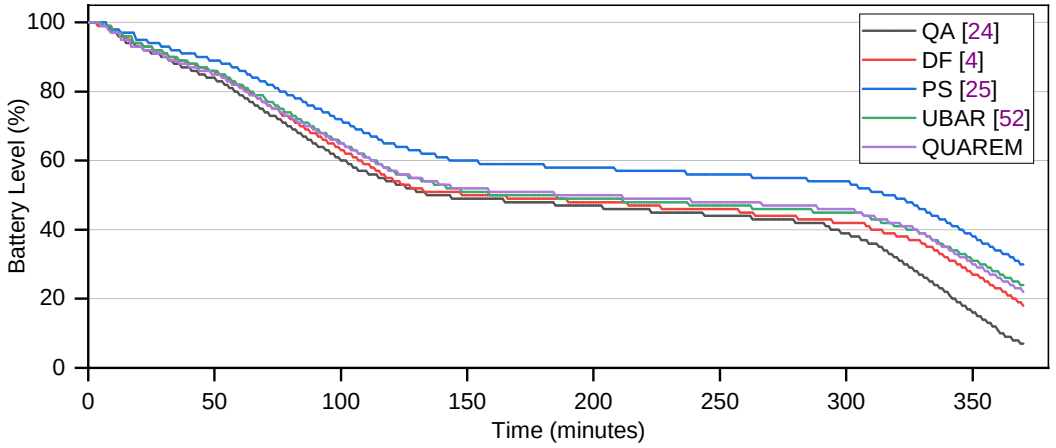

Fig. 11. Battery discharge profile for the different approaches considered.

the expected battery life goal due to its initial higher QoS target and also being agnostic to the battery life goal. QUAREM on the other hand is able to meet the battery life expectation of the user at 80% initial battery level, however, it fails at 70%, which is also the same initial battery level at which PS approach fails. This shows QUAREM's ability to compare favourably, with power saving techniques. The secondary Y-axis of Fig. 12 shows the final battery level for the rest of the initial battery levels considered. The figure reveals that while QA, DF and UBAR approaches run out of battery with slight variation in user's usage behaviour (reduced battery from the start), QUAREM was able to adjust resource allocation to meet battery life by up to 25%.

Fig. 12 also shows the percentage of time each approach was active (primary Y-axis) at different initial battery level. It can be seen that PS approach has the highest percentage of time active because it is focused on power saving irrespective of the impact on the users' QoE. QUAREM was able to provide higher active time in comparison to UBAR, DF and QA approaches while maximising the QoE. Indeed, PS has better battery discharge and active time profiles than the approaches considered in Figs. 11 and 12. This is because the PS approach is designed to minimise the power consumption even at the expense of QoS, thus negatively affecting the QoE. On the other hand, QUAREM is aimed at maximising the user's QoE across the battery discharge by adaptively managing the QoS and the energy consumption. The idea is to provide a satisfactory service across a battery-life goal. For example, Fig. 11 shows PS with a 30% battery level (remaining energy) at the plug-in time, while QUAREM has a 22% battery level at the same plug-in time. However, it will be worthless to have 30% or more battery level at the plug-in time when the user is dissatisfied with or even abandoning the services provided for most time of the day (PS in Fig. 11); QUAREM tries to balance the QoS while focusing on the battery life goal, thus having a slightly lower battery level (22%) at the plug-in time but the user is satisfied with the services throughout the day. This also shows the benefit of resource management with the knowledge of the battery-life goal as against generic energy saving approaches. Though the focus of QUAREM is the short-term battery discharge (time from plug-out to the next plug-in), its impact on longer-term battery life (i.e., the number of charge/discharge cycles before the battery fails to operate satisfactorily) cannot be overemphasized since consistent good short term battery life management like the one proposed in this work that helps in maintaining the SOC level for a long time and also alleviating the on-chip

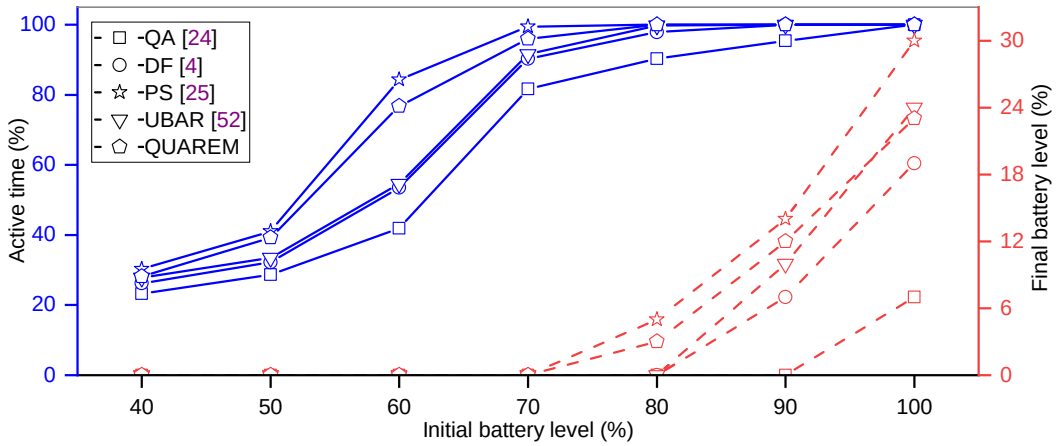

Fig. 12. Summary of the percentage of the final battery level and the percentage of time each approach was active at different initial battery level.

thermal profile through capping off the excessive heat that might be generated as a result of unnecessary high frequency will help in improving the overall long battery life.

5.3 QoE Improvement

To evaluate the benefit of *QUAREM* in meeting the user's battery life expectancy with little or no QoS degradation, i.e. enhancing the QoE, we employ the model utilised in the works of [19, 33] to quantify the QoE, an exponential relationship between QoS and QoE as earlier stated in Section 2. We also compare the results obtained by different state-of-the-art approaches using the commercial mobile smartphone and usage pattern described in Section 4.

Fig. 13 shows the comparison of instantaneous QoE for five resource management techniques. The values are computed using the exponential QoS to QoE relationship [19] from the frame jank rate (QoS) obtained every second while running the considered scenario in the device. The higher the value of instantaneous QoE the better it is because it shows less/fewer janks, thus, giving rise to a smoother and better experience. Fig. 13 reveals that QA, DF, UBAR and QUAREM approaches have relatively higher instantaneous QoE with an occasional dip at certain instances. This dip is mostly as a result of the resource requirement and adjustment associated with loading and starting up an application, which is accustomed to fluctuation in the instantaneous QoE. For example, this can be observed at time t = 220s, 1000s, for the QUAREM approach. Similarly, identical fluctuations can be noticed with other approaches as well. QUAREM approach closely compares with the QA technique with a slightly higher instantaneous QoE values compared to DF approach. UBAR also has comparatively high QoE at the beginning when the battery level is high, but decreases as the battery level decreases due to the approach's reduction in target QoS when a plug-in is not predicted and the battery level is below a certain threshold (70 % and 50 %). Overall, it has a higher QoE compared to PS approach. PS on the other hand, shows a relatively low instantaneous QoE with dips that goes below 40 % indicating significantly higher janks (>60 %). This is because of the low frequency of operation that is mostly employed by the PS method. Table 4 summarises the variation improvement of the instantaneous QoE of the QUAREM approach against the existing approaches. From the table, the QA approach has maximum and minimum QoE of 100% and 60.9% respectively with variance of 86.5 and a standard deviation of 9.30. Similarly, DF, UBAR and QUAREM have the

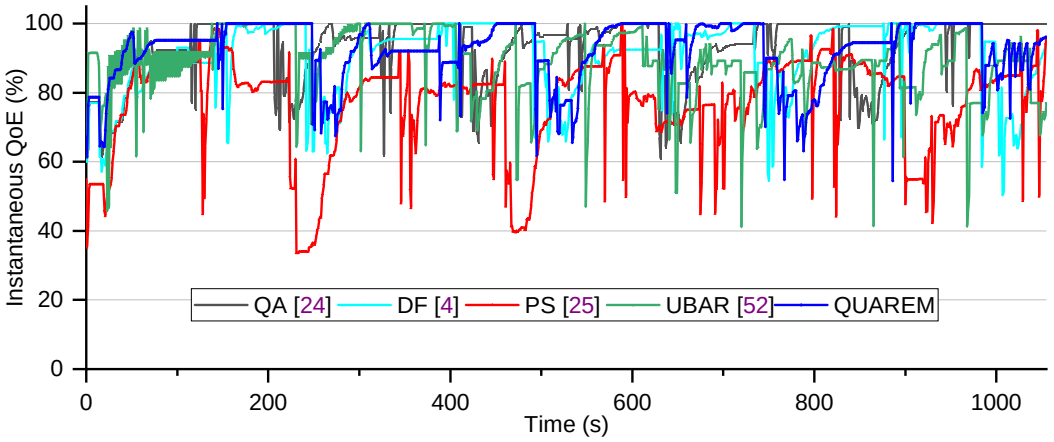

Fig. 13. Comparison of instantaneous QoE for five different resource management techniques (higher is better) while running Google Maps and YouTube repeatedly. The QoS is measured as frames rate jank at every second.

same maximum QoE of 100% and minimum values of 50.2%, 41.3% and 54.6% respectively. While DF has a variance and standard deviation of 105 and 10.3 respectively, UBAR and QUAREM have variance of 91.7 and 76.7 with standard deviation of 9.57 and 8.76 respectively. PS on the other hand has 99.8% and 33.5% for maximum and minimum values respectively with variance and standard deviation of 200 and 14.2, respectively.

In summary, QUAREM approach shows a lower variance and standard deviation compared to the considered techniques with percentage variation improvement (statistical variance) of 11.3%, 16.4%, 27.1% and 61.8% in comparison to QA, UBAR, DF and PS respectively (as shown in Table 4). This significantly improves the overall users' QoE because higher variation means inconsistent usage experience thus, affecting the overall QoE.

Fig. 14 shows the averaged QoE comparison results for the five different resource management approaches while starting the day at different percentage of initial battery level. The seven different initial battery level are evaluated with the same usage scenario as described in Section 4.2.2. The average QoE in Fig. 14 is computed by taking the average of the instantaneous QoE values obtained in Fig. 13 for the entire usage scenario with a penalty of *0* when the battery is fully depleted to show user's dissatisfaction. QA shows higher average QoE at 100 % initial battery level but declines at subsequent battery levels, 90 % through 40 % due to its inability to meet up with the battery

Table 4. Variation improvement of the instantaneous QoE of the QUAREM approach against the existing approaches while running Google Maps and YouTube applications.

Approach	Max	Min	Mean	Variance	STD	% of Variation Compared to QUAREM
QA [24]	100	60	93.3	86.5	9.30	11.3 %
DF [4]	100	50.2	91.2	105	10.3	27.1%
PS [25]	99.8	33.5	77.1	201	14.2	61.8%
UBAR [52]	100	41.3	89.3	91.7	9.57	16.4%
QUAREM	100	54.6	92.3	76.7	8.76	0

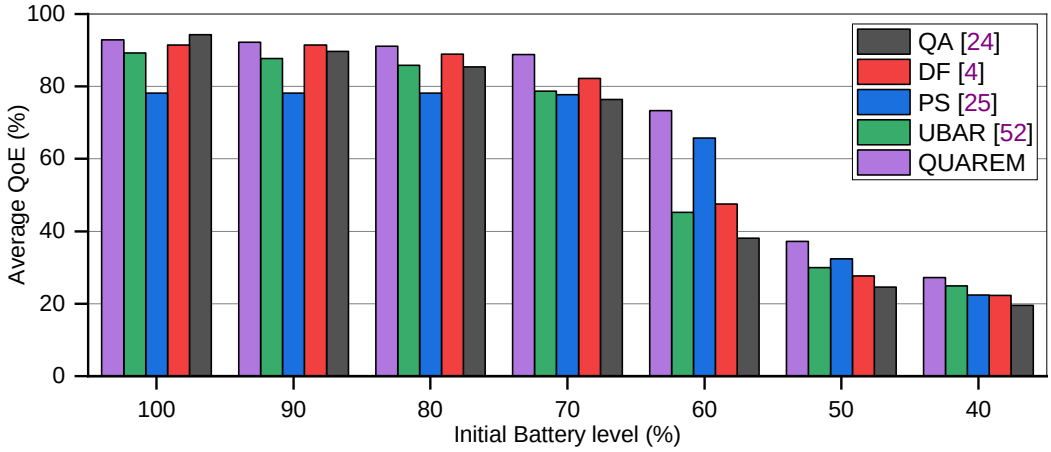

Fig. 14. Comparison of average QoE for five different resource management techniques while starting the day at different initial battery level (%).

life expectation of the user. Similarly, DF shows a considerable high average QoE at 100% and 90% battery levels but it also declines as the initial battery levels considered are lowered. On the other hand, QUAREM shows a comparatively higher average QoE compared to other techniques due to its ability to adaptively manage the resources to meet the battery life expectation of the user. UBAR was unable to provide higher averaged QoE at 100% initial battery level compared to DF, QUAREM and QA approaches due to over-constrained DVFS despite the fact that the energy (battery) is sufficient for the expected battery life. It is also able to provide higher averaged QoE compared to QA and DF approaches due to its energy saving ability at low SOC, thus, sustaining higher expected battery life compared to DF and QA. Though even QUAREM could not guarantee the battery life at initial battery level beyond 70% as shown in Fig. 12, QUAREM is able to maintain a higher average QoE for almost all the situations. PS maintained a relatively low average QoE compared to the other approaches when starting the day at battery level >70%.

In summary, Table 5 depicts the QoE improvement that can be obtained using QUAREM in comparison to existing approaches. The result demonstrates that our approach can achieve up to 92.6% the improvement compared to QA that aims at maximising QoS and up to 54.1% compared to the DF that is with energy awareness. It also shows that QUAREM achieves 4.05% to 62.2% QoE improvement when compared with UBAR approach. This is because UBAR often fails in meeting

Table 5. Summary of QoE improvement of QUAREM compared to existing approaches at different percentage of initial battery level (IBL).

Approach	At 100% IBL	At 90% IBL	At 80% IBL	At 70% IBL	At 60% IBL	At 50% IBL	At 40% IBL
QA [24]	-1.51%	2.87%	6.65%	16.2%	92.6%	51.6%	39.2%
DF [4]	1.57%	0.82%	2.50%	8.01%	54.1%	34.4%	22.4%
PS [25]	18.8%	17.9%	16.5%	14.1%	11.5%	15.1%	21.8%
UBAR [52]	4.05%	5.17%	6.09%	12.8%	62.2%	24.3%	9.44%

*IBL = Initial Battery Level

the user's battery life expectation with slight variation in user's usage pattern and/or reduced battery from the start, but QUAREM is able to adapt with up to 25% variation with little or no QoS degradation, thus improving the overall QoE. This further demonstrates the benefit of considering both the user desired plug-in time and the energy requirement in maximising QoE. For a new user and platform whose data does not exist a priori, the QUAREM approach is turned off on the first day while the user's data is profiled using the default runtime management. Subsequently, the previously profiled data is used for the current day while the model is updated with the current data and continuously adapt based on the usage pattern of the user.

5.3.1 Effect of Varying the Penalty. We also analysed the effect of varying the penalty to show users' dissatisfaction if the battery life expectation is missed when employing the QUAREM and existing approaches. This is considered because existing approaches that utilised the same IQX model for the QoE computation applied a penalty of -100 when the battery life expectation is missed; for example, Lee *et al*. [33]. Table 6 shows average QoE results when applying a penalty of −50 and −100 at varying initial battery levels. With a higher penalty value, e.g., −100, the QA, DF, and UBAR approaches either have nominal average QoE value (1.22) or zero average QoE value (capped to zero to avoid negative values), but QUAREM maintains a high average QoE (52.6). Similarly, when the approaches have a zero average QoE value at a penalty of −50, QUAREM still has a significant average QoE value. However, with a further decrease in the value of the initial battery level, e.g., 40% and beyond, all the approaches have a zero average QoE value. It can also be observed that QUAREM provides a much higher average QoE (higher than even the values in

Table 6. Summary of the average QoE at a varying penalty for QUAREM and the existing approaches

Approach		At 100% IBL	At 90% IBL	At 80% IBL	At 70% IBL	At 60% IBL	At 50% IBL	At 40% IBL
QA [24]	-50	94.3 (-1.5%)	87.4 (5.5%)	80.6 (13%)	67.3 (29%)	9.09 (593%)	0 (0%)	0 (0%)
	-100	94.3 (-1.5%)	85.1 (8.3%)	75.7 (20.3%)	58.1 (45.8%)	0 (0%)	0 (0%)	0 (0%)
DF [4]	-50	91.5 (1.5%)	91.5 (0.8%)	87.5 (4.1%)	77.3 (12.3%)	24.4 (158.2%)	0 (0%)	0 (0%)
	-100	91.5 (1.5%)	91.5 (0.8%)	86.2 (5.7%)	72.5 (16.8%)	1.22 (4211%)	0 (0%)	0 (0%)
PS [25]	-50	78.2 (18.8%)	78.2 (17.9%)	78.2 (16.5%)	77.5 (12%)	58.0 (8.6%)	2.82 (159%)	0 (0%)
	-100	78.2 (18.8%)	78.2 (17.9%)	78.2 (16.5%)	77.2 (9.7%)	50.1 (5%)	0 (0%)	0 (0%)
UBAR [52]	-50	89.3 (4%)	87.7 (5.1%)	85.8 (6.2%)	74.5 (16.5%)	22.6 (179%)	0 (0%)	0 (0%)
	-100	89.3 (4%)	87.7 (5.1%)	85.6 (6.4%)	70.4 (20.3%)	0 (0%)	0 (0%)	0 (0%)
QUAREM	-50	92.9 (0%)	92.2 (0%)	91.1 (0%)	86.8 (0%)	63.0 (0%)	7.3 (0%)	0 (0%)
	-100	92.9 (0%)	92.2 (0%)	91.1 (0%)	84.7 (0%)	52.6 (0%)	0 (0%)	0 (0%)

*IBL = Initial Battery Level

Table 5) over existing approaches. This indicates that if you penalise failure to meet battery life expectations more, all existing approaches fall over sooner, thus, more remarkable improvement for QUAREM, which ensures users' battery life expectations are met within the QoS requirement.

Although a dataset of this size is typical in this literature, we have proposed a competitive approach rather than extending the evaluations of existing ones. We have also evaluated our approach similarly to existing works and have shown improved results, and scaling the evaluation is the top of our future work. This will involve getting more diverse user groups to evaluate the data.

5.4 Adaptability and Scalability

This section discusses the adaptability of our approach with respect to users with erratic (multiple) charging behaviour and its scalability to other portable electronics, both hardware and software.

5.4.1 Users with erratic charging patterns. Though the QUAREM approach was designed to maximise the user's QoE across the battery discharge for users with regular charging cycle as discussed in Section 2, we also selected five (5) individual users from the Sherlock dataset with erratic (multiple) charging patterns to evaluate the effectiveness of QUAREM for different charging patterns. Among the users with erratic behaviour randomly selected include two (2) users that plug-in around 15:00 of the day in addition to their plug-in times at the end of the day. Similarly, three (3) users that plug-in twice between their normal plug-in times i.e. around 12:00 and 16:00 in addition to their normal plug-in times at the end of the day were also considered.

Fig. 15 shows the average QoE for the different resource management approaches while considering these erratic charging patterns at different initial battery levels (%). In the figure, QA, DF, and PS show consistent values with QA having a higher average QoE across all the initial battery levels considered. This is because these approaches have no specific battery life goal to meet and with an erratic charging pattern, the device is never expected to span the entire day but is often plugged intermittently within the day. However, for a larger proportion of the users that do not exhibit erratic charging patterns, these approaches tend to be worst due to their inability to guarantee battery life across the user's expected battery life (charging cycle) as shown in Figs. 12 and 14. On the other hand, QUAREM and UBAR approaches that work based on plug-in times prediction and

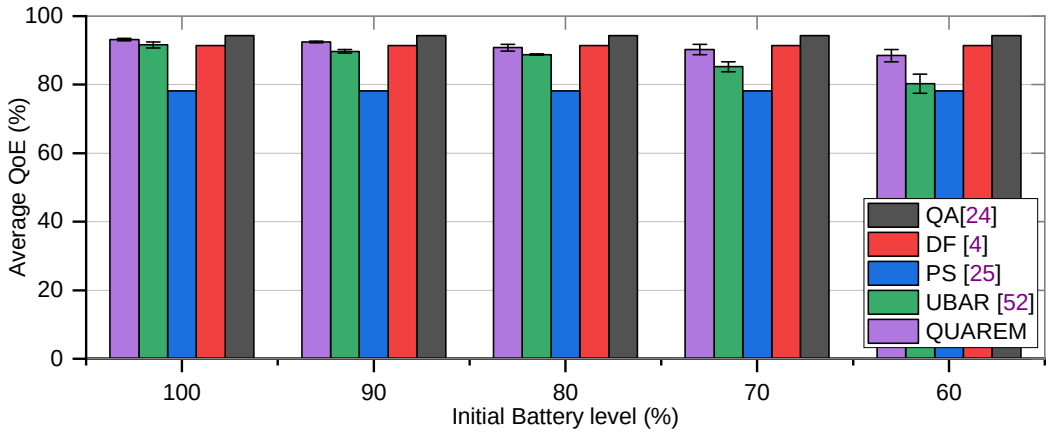

Fig. 15. Comparison of average QoE for five different resource management techniques while considering erratic charging patterns and starting the day at different initial battery level (%).

battery life expectation behave differently from these approaches. While QUAREM was able to adapt very well with a higher average QoE value compared to DF, UBAR and PS at 100 % and 90 % initial battery levels, it is slightly outperformed by the DF approach at 80 %, 70 % and 60 % initial battery levels due to the impact of the erratic user behaviour on the resource management strategy. This is because, at these initial battery levels (80 %, 70 % and 60 %), the energy demand is predicted to be greater than the remaining energy for the predicted plug-in time, and thus, the QUAREM approach enforces frugal resource management to ensure that the remaining energy meets the expected battery life of the user. However, at midday or anytime within the day (15:00, 12:00 and 16:00) when the user randomly plugs in the device to charge, that extra energy is considered and optimal management (optimum cap, ϕ) is enforced since the energy demand prediction is done at each time step. Meanwhile, the plug-in time will not be affected because the approach is configured to acknowledge plug-in at the end of the day rather than multiple charging within the day. In contrast, UBAR shows a lower average QoE compared to QA, DF and QUAREM approaches for the scenario considered, however, if the random charging was done at SOC level > 70 %, UBAR outperforms DF approach. This is because the strategy involves relaxing the QoE whenever the SOC is < 70 % and plug-in is not predicted, which in this case occurs randomly, and thus, hampering the prediction accuracy.

In summary, despite the erratic charging pattern, QUAREM is able to maintain a higher QoE value > 88 %, which outperforms UBAR and PS in all the scenarios. This indicates that QUAREM can adapt with comparable average QoE despite erratic user behaviour.

5.4.2 Scalability to other portable electronics (hardware and software). Although smartphone is the platform that QUAREM was experimented on, there is no reason this would not work on laptops, tablets and other general-purpose mobile computing platforms since, in principle, we need a device with a battery whose discharge is variable based on the user behaviour and input, and some ability to control the power consumption dynamically. Also, some specialise devices like Kindle and battery-powered smartwatches will still be applicable. But when it comes to specialised electronic devices that do not meet these requirements because they are carefully designed to optimise energy consumption for a specific task, QUAREM might not scale.

The OS provides the support required for the control; as long as it provides those functionalities, it is scalable. The more heterogeneous the system is, the more complex the problem becomes, and more dimensions are for managing it, which our approach will also apply. Even though different memory architectures will impact the power savings, it is still applicable unless there is a change in the relationship between the CPU frequency and the power consumption. Likewise, the approach will also scale with single or multi-core architectures without GPU.

5.5 Overhead

5.5.1 Runtime overhead. The runtime overhead of the proposed QUAREM approach depends on the duration of the plug-in time and the time taken to complete computations such as updating the model and reconfiguring of the DVFS settings at each time step before the next plug-in activity. Since the model is less compute-intensive as it is only updated with the most recent value for the prediction, the computation and configuration are done quite quickly. At each time step, on average, updating the model, the predictions and DVFS reconfiguration take 400 ms. Considering the entire plug-in time of ≈ 15 h on average with 180-time steps (5 min interval), the total timing overhead is 72 s, which is only 0.13 % of the entire plug-in time. The entire implementation has small memory footprint with less than 1% computational overhead.

We also observed the CPU utilisation with the help of Snapdragon profiler [32], which is a tool that allows system information such as power, processing elements, memory, thermal and network,

to be captured and analysed in real-time. We monitor the system usage with and without our approach running to determine the overhead. We observed an average of less than 0.5 % increase in total CPU utilisation, which is also quite low. This is because the firmware is allowed to manage the frequency switching, thus making the switching overhead, not more than usual. This shows that QUAREM can be efficiently used to maximise QoE while meeting the user's expected battery life.

For the energy consumption, an exact value can not be given because the experimental results measure the actual energy consumption, and therefore any overheads are inherently included. However, this is < 1% of the entire battery usage until plug-in as it is not even captured by the power monitoring system within the mobile phone. Comparing these results with existing work such as UBAR [52], QUAREM compares favourably as similar results are obtained with improved average QoE.

5.5.2 Offline overhead. The offline profiling time consists of the times taken to determine the optimum frequency cap, ϕ, and the various efficient frequency caps, ϵ, for the range of δ values to achieve the desired user's battery life expectation. This time depends upon the number of iterations considered and the time for the productive task of an application to be completed, which mostly varies. In our case, we consider four popular applications, which takes $\approx$ 68–120 s for a productive task to be completed for each application. We also considered two-step size frequency starting slightly below the mid-frequency up through to the highest frequency of the CPU core with $\approx$ 150 runs (iterations) per application. The whole offline profiling takes $\approx$ 146 *hours* ($\approx$ 6 *days*), which is given by: *no. of applications* $\times$ *average time taken* $\times$ *no. of runs* $\times$ *no. of different frequencies*. Since it is a one-time process whose benefit leads to maximization of user's QoE as earlier shown, it is suitable to be employed.

6 CONCLUSIONS

This paper has explored a QoE-aware adaptive resource management technique, *QUAREM*, for mobile MPSoC platforms. Firstly, a study on user energy usage pattern and plug-in time is conducted. The analysis revealed wide variation in plug-in times and usage patterns by users. Thus, a prediction model was built and used to maximise the QoE by balancing the QoS and the user desired battery life. Experimental evaluation on Google pixel 3 smartphone for different initial battery levels has shown that our approach can be used to achieve improved battery life and QoE. The results also show that QUAREM achieves averaged QoE improvement when compared with the state-of-the-art focusing on maximising QoE, and thus QUAREM illustrates the benefit of considering user plug-in time and energy requirement in maximising QoE. Our future work will involve getting more diverse user groups to evaluate our data and testing the app with a diverse set of live users. We will also consider tuning the hyper-parameters online to factor in significant differences in usage patterns (such as vacation, breaks, etc.) over time or at different times in the year.

ACKNOWLEDGMENTS

This study was supported by Petroleum Technology Development Fund (PTDF) with grant number 1526/19. All data supporting this study are openly available from the University of Southampton repository at https://doi.org/10.5258/SOTON/D2153

REFERENCES

[1] Mustafa Imran Ali, Bashir M. Al-Hashimi, Joaquín Recas, and David Atienza. 2010. Evaluation and Design Exploration of Solar Harvested-Energy Prediction Algorithm. In *Proceedings of the Conference on Design, Automation and Test in Europe* (Dresden, Germany) *(DATE '10)*. European Design and Automation Association, Leuven, BEL, 142–147.

[2] Apkpure.com. 2015. Funf Journal. Retrieved February 3, 2020 from https://apkpure.com/funf-journal/edu.mit.media.funf.journal

[3] ARM. 2018. Welcome to Documentation for Workload Automation. Retrieved April 4, 2020 from https://workload-automation.readthedocs.io/en/latest/index.html

[4] ARMDeveloper. 2019. Energy Aware Scheduling (EAS). Retrieved March 2, 2020 from https://developer.arm.com/tools-and-software/open-source-software/linux-kernel/energy-aware-scheduling

[5] Martin Armstrong. 2020. The Apps Americans Can't Live Without. Retrieved February 10, 2021 from https://www.statista.com/chart/23230/apps-people-cant-do-without-united-states/

[6] Holst Arne. 2020. Number of Smartphone users Worldwide from 2016 to 2023. Retrieved March 22, 2020 from https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide

[7] Nilanjan Banerjee, Ahmad Rahmati, Mark D. Corner, Sami Rollins, and Lin Zhong. 2007. Users and Batteries: Interactions and Adaptive Energy Management in Mobile Systems. In *Proceedings of the 9th International Conference on Ubiquitous Computing* (Innsbruck, Austria) *(UbiComp '07)*. Springer-Verlag, Berlin, Heidelberg, 217–234.

[8] James RB Bantock, Bashir M Al-Hashimi, and Geoff V Merrett. 2020. Mitigating Interactive Performance Degradation from Mobile Device Thermal Throttling. *IEEE Embedded Systems Letters* (2020).

[9] James RB Bantock, Vasileios Tenentes, Bashir M Al-Hashimi, and Geoff V Merrett. 2017. Online tuning of dynamic power management for efficient execution of interactive workloads. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 1–6.

[10] Karunakar R Basireddy, Amit Kumar Singh, Bashir M Al-Hashimi, and Geoff V Merrett. 2019. AdaMD: Adaptive Mapping and DVFS for Energy-Efficient Heterogeneous Multicores. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2019), 2206–2217.

[11] Sascha Bischoff, Andreas Hansson, and Bashir M Al-Hashimi. 2013. Applying of quality of experience to system optimisation. In *2013 23rd International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. IEEE, 91–98.

[12] Xiang Chen, Yiran Chen, Zhan Ma, and Felix C. A. Fernandes. 2013. How is Energy Consumed in Smartphone Display Applications?. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications* (Jekyll Island, Georgia) *(HotMobile '13)*. Association for Computing Machinery, New York, NY, USA, Article 3, 6 pages. https://doi.org/10.1145/2444776.2444781

[13] Yonghun Choi, Seonghoon Park, and Hojung Cha. 2019. Optimizing Energy Efficiency of Browsers in Energy-Aware Scheduling-Enabled Mobile Devices. In *The 25th Annual International Conference on Mobile Computing and Networking* (Los Cabos, Mexico) *(MobiCom '19)*. Association for Computing Machinery, New York, NY, USA, Article 48, 16 pages. https://doi.org/10.1145/3300061.3345449

[14] Android Developer. 2021. Android 8.0 Behavior Changes. Retrieved March 15, 2021 from https://developer.android.com/about/versions/oreo/android-8.0-changes#all-apps

[15] Somdip Dey, Amit Kumar Singh, Xiaohang Wang, and Klaus McDonald-Maier. 2020. User interaction aware reinforcement learning for power and thermal efficiency of CPU-GPU mobile MPSoCs. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1728–1733.

[16] Bryan Donyanavard, Tiago Mück, Santanu Sarma, and Nikil Dutt. 2016. SPARTA: Runtime task allocation for energy efficient heterogeneous manycores. In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 1–10.

[17] Ismat Chaib Draa, Fabien Bouquillon, Smail Niar, and Emmanuelle Grislin-Le Strugeon. 2019. Machine learning for improving mobile user satisfaction. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. 1200–1207.

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

[19] Markus Fiedler, Tobias Hossfeld, and Phuoc Tran-Gia. 2010. A generic quantitative relationship between quality of experience and quality of service. *IEEE Network* 24, 2 (2010), 36–41.

[20] Andrei Frumusanu. 2018. The Google Pixel 3 Review: The Ultimate Camera test. Retrieved August 4, 2020 from https://www.anandtech.com/show/13474/the-google-pixel-3-review

[21] Andrei Frumusanu. 2018. Improving the Exynos 9810 Galaxy S9: Part 2 - catching up with the Snapdragon. Retrieved March 27, 2021 from https://www.anandtech.com/show/12620/improving-the-exynos-9810-galaxy-s9-part-2

[22] Benjamin Gaudette, Carole-Jean Wu, and Sarma Vrudhula. 2016. Improving smartphone user experience by balancing performance and energy with probabilistic QoS guarantee. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE Computer Society, 52–63. https://doi.org/10.1109/HPCA.2016.7446053

[23] Marco ET Gerards, Johann L Hurink, and Jan Kuper. 2014. On the interplay between global DVFS and scheduling tasks with precedence constraints. *IEEE Trans. Comput.* 64, 6 (2014), 1742–1754.

[24] Google Git. 2020. Android CPUFreq Governors. Retrieved January 13, 2021 from https://android.googlesource.com/kernel/msm/+/refs/tags/android-11.0.0_r0.43/Documentation/cpu-freq/governors.txt

[25] Utkarsh Goel, Stephen Ludin, and Moritz Steiner. 2020. Web Performance with Android's Battery-Saver Mode. *arXiv preprint arXiv:2003.06477* (2020).
[26] Andrés Goens, Robert Khasanov, Jeronimo Castrillon, Marcus Hähnel, Till Smejkal, and Hermann Härtig. 2017. Tetris: a multi-application run-time system for predictable execution of static mappings. In *Proceedings of the 20th International Workshop on Software and Compilers for Embedded Systems*. 11–20.
[27] Rob J Hyndman and George Athanasopoulos. 2018. *Forecasting: principles and practice* (2nd ed.). OTexts, Australia.
[28] Selim Ickin, Katarzyna Wac, Markus Fiedler, Lucjan Janowski, Jin-Hyuk Hong, and Anind K Dey. 2012. Factors influencing quality of experience of commonly used mobile applications. *IEEE Communications Magazine* 50, 4 (2012), 48–56.
[29] Samuel Isuwa, Somdip Dey, Amit Kumar Singh, and Klaus McDonald-Maier. 2019. Teem: Online thermal-and energy-efficiency management on cpu-gpu mpsocs. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 438–443.
[30] 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 *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.
[31] 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*. ACM, New York, NY, USA, 1–6. https://doi.org/10.1145/3195970.3195994
[32] Sei Ping Lau, Alex S Weddell, Geoff V Merrett, and Neil M White. 2014. Energy-neutral solar-powered street lighting with predictive and adaptive behaviour. In *Proceedings of the 2nd International Workshop on Energy Neutral Sensing Systems*. 13–18.
[33] 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 *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 64–69.
[34] Xueliang Li, Guihai Yan, Yinhe Han, and Xiaowei Li. 2013. SmartCap: User Experience-Oriented Power Adaptation for Smartphone's Application Processor. In *Proceedings of the Conference on Design, Automation and Test in Europe* (Grenoble, France) *(DATE '13)*. EDA Consortium, San Jose, CA, USA, 57–60.
[35] Sumit K Mandal, Ganapati Bhat, Janardhan Rao Doppa, Partha Pratim Pande, and Umit Y Ogras. 2020. An energy-aware online learning framework for resource management in heterogeneous platforms. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 25, 3 (2020), 1–26.
[36] Sumit K Mandal, Ganapati Bhat, Chetan Arvind Patil, Janardhan Rao Doppa, Partha Pratim Pande, and Umit Y Ogras. 2019. Dynamic resource management of heterogeneous mobile platforms via imitation learning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 12 (2019), 2842–2854.
[37] Youssef Mansour, Hamdan Hammad, Omnia Abu Waraga, and Manar Abu Talib. 2021. Energy Management Systems and Smart Phones: A Systematic Literature Survey. In *2021 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)*. IEEE, 1–7.
[38] Yisroel Mirsky, Asaf Shabtai, Lior Rokach, Bracha Shapira, and Yuval Elovici. 2016. Sherlock vs moriarty: A smartphone dataset for cybersecurity research. In *Proceedings of the 2016 ACM workshop on Artificial intelligence and security*. 1–12.
[39] Tulika Mitra. 2015. Heterogeneous Multi-core Architectures. *Information and Media Technologies* 10, 3 (2015), 383–394. https://doi.org/10.11185/imt.10.383
[40] Sebastian Möller and Alexander Raake. 2014. *Quality of experience: advanced concepts, applications and methods*. Springer.
[41] S O'Dea. 2020. Smartphone unit shipments by price category worldwide from 2012 to 2022. Retrieved December 02, 2021 from https://www.statista.com/statistics/934471/smartphone-shipments-by-price-category-worldwide/
[42] Dhinakaran Pandiyan and Carole-Jean Wu. 2014. Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 171–180.
[43] Jurn-Gyu Park, Nikil Dutt, Hoyeonjiki Kim, and Sung-Soo Lim. 2016. HiCAP: Hierarchical FSM-Based Dynamic Integrated CPU-GPU Frequency Capping Governor for Energy-Efficient Mobile Gaming. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design* (San Francisco Airport, CA, USA) *(ISLPED '16)*. Association for Computing Machinery, New York, NY, USA, 218–223. https://doi.org/10.1145/2934583.2934588
[44] Jurn-Gyu Park, Chen-Ying Hsieh, Nikil Dutt, and Sung-Soo Lim. 2016. Co-Cap: Energy-Efficient Cooperative CPU-GPU Frequency Capping for Mobile Games. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing* (Pisa, Italy) *(SAC '16)*. Association for Computing Machinery, New York, NY, USA, 1717–1723. https://doi.org/10.1145/2851613.2851671
[45] Alok Prakash, Siqi Wang, Alexandru Eugen Irimiea, and Tulika Mitra. 2015. Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms. In *2015 33rd IEEE International Conference on Computer Design (ICCD)*.

IEEE, 208–215.

[46] Pijush Kanti Dutta Pramanik, Nilanjan Sinhababu, Bulbul Mukherjee, Sanjeevikumar Padmanaban, Aranyak Maity, Bijoy Kumar Upadhyaya, Jens Bo Holm-Nielsen, and Prasenjit Choudhury. 2019. Power consumption analysis, measurement, management, and issues: a state-of-the-art review of smartphone battery and energy usage. *IEEE Access* 7 (2019), 182113–182172.

[47] Qualcomm.com. 2017. Snapdragon 845 Mobile Platform. Retrieved June 2, 2020 from https://www.qualcomm.com/products/snapdragon-845-mobile-platform

[48] Vijay Janapa Reddi, Hongil Yoon, and Allan Knies. 2018. Two billion devices and counting. *IEEE Micro* 38, 1 (2018), 6–21.

[49] Basireddy Karunakar Reddy, Amit Kumar Singh, Dwaipayan Biswas, Geoff V Merrett, and Bashir M Al-Hashimi. 2017. Inter-cluster thread-to-core mapping and DVFS on heterogeneous multi-cores. *IEEE Transactions on Multi-Scale Computing Systems* 4, 3 (2017), 369–382.

[50] Krishna Sekar. 2013. Power and thermal challenges in mobile devices. In *Proceedings of the 19th annual international conference on Mobile computing & networking*. 363–368.

[51] 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 *2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID)*. IEEE, 43–48. https://doi.org/10.1109/VLSID49098.2020.00025

[52] 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

[53] Amit Kumar Singh, Alok Prakash, Karunakar Reddy Basireddy, Geoff V. Merrett, and Bashir M. Al-Hashimi. 2017. Energy-Efficient Run-Time Mapping and Thread Partitioning of Concurrent OpenCL Applications on CPU-GPU MPSoCs. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 147 (Sept. 2017), 22 pages. https://doi.org/10.1145/3126548

[54] Robert Triggs. 2017. Everything you need to know about Qualcomm's Snapdragon 845. https://www.androidauthority.com/qualcomm-snapdragon-845-specs-820561/.

[55] W. Waag and D.U. Sauer. 2009. SECONDARY BATTERIES – LEAD– ACID SYSTEMS | State-of-Charge/Health. In *Encyclopedia of Electrochemical Power Sources*, Jürgen Garche (Ed.). Elsevier, Amsterdam, 793–804. https://doi.org/10.1016/B978-044452745-5.00149-0

[56] Rafael J. Wysocki. 2017. Intel pstate CPU Performance Scaling Driver. Retrieved January 10, 2021 from https://www.kernel.org/doc/html/v4.12/admin-guide/pm/intel_pstate.html

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

[58] Kaige Yan, Xingyao Zhang, Jingweijia Tan, and Xin Fu. 2016. Redefining QoS and Customizing the Power Management Policy to Satisfy Individual Mobile Users. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture* (Taipei, Taiwan) *(MICRO-49)*. IEEE Press, Article 53, 12 pages.

[59] Yuhao Zhu, Matthew Halpern, and Vijay Janapa Reddi. 2015. Event-based scheduling for energy-efficient QoS (eQoS) in mobile Web applications. In *21st IEEE International Symposium on High Performance Computer Architecture, HPCA 2015, Burlingame, CA, USA, February 7-11, 2015*. IEEE Computer Society, 137–149. https://doi.org/10.1109/HPCA.2015.7056028