Received March 19, 2020, accepted April 3, 2020, date of publication April 7, 2020, date of current version April 23, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2986214

Proactive Scenario Characteristic-Aware Online Power Management on Mobile Systems

SODAM HAN[1], (Student Member, IEEE), YONGHEE YUN[2], YOUNG HWAN KIM[1], (Senior Member, IEEE), AND SEOKHYEONG KANG[1], (Member, IEEE)

[1]Department of Electronic and Electrical Engineering, Pohang University of Science and Technology, Pohang 37673, South Korea
[2]System LSI Business, Samsung Electronics Company Ltd., Hwaseong 18448, South Korea

Corresponding author: Seokhyeong Kang (shkang@postech.ac.kr)

This work was supported in part by Samsung Electronics Company, Ltd., and in part by the Ministry of Science and ICT (MSIT), South Korea, through the ICT Consilience Creative Program, supervised by the Institute for Information and Communications Technology Planning and Evaluation (IITP), under Grant IITP-2019-2011-1-00783.

ABSTRACT Modern mobile systems are requested to execute diverse user scenarios. Depending on the types of user scenarios, mobile systems utilize hardware resources differently. Thus, power management policies of mobile systems must adapt to various user scenarios. In this paper, we propose a dynamic voltage/frequency scaling (DVFS) policy to increase the energy efficiency of multicore mobile systems by adapting to user scenarios. The proposed policy provides effective power management regardless of user scenarios by using operation characteristics that can represent the execution behavior of various user scenarios. Furthermore, the proposed policy is suitable for modern mobile systems in which online power management is essential, because it does not require preliminary knowledge of target scenarios. To balance the trade-off between energy consumption and quality-of-service (QoS), the proposed scenario-aware policy provides 'just enough' processing speed to process the requested amount of work at the given parallelism level. To demonstrate the practicality of the proposed policy, we evaluated the effectiveness of the proposed scenario-aware policy for real-world user scenarios. Compared to the conventional DVFS policies, the proposed scenario-aware policy achieved a maximum of 25.5 % energy saving on the mobile system that uses asymmetric multicore CPU, and a maximum of 30.7 % energy saving on the mobile system that uses symmetric multicore CPU, without any QoS violation that degrades user experiences.

INDEX TERMS Dynamic voltage/frequency scaling (DVFS), energy efficiency, quality-of-service (QoS), workload-aware power management, application-aware power management.

I. INTRODUCTION

Nowadays, the usage of mobile devices, such as smartphones and tablets, is not limited to communication. End users use modern mobile devices to execute various user scenarios beyond communication, such as web browsing, social networking, multimedia streaming, and gaming. In this context, mobile devices require high processing speed without degrading user satisfaction. Thus, modern mobile devices generally adopt high-performance processors such as CPU that has asymmetric multicore architecture (e.g., big.LITTLE technology [1]), GPU, and AI accelerator.

Integration of high-performance processors and multiple IPs increases quality-of-service (QoS) of mobile devices but decreases power budget allocated to each component due to the limited capacity of batteries. Consequently, the decrease in power budget for each component shortens the entire battery life of mobile devices [2]. Furthermore, as the size of mobile devices is scaling down, their power density increases. High power density can cause thermal hot spots and overheating of devices; these problems can degrade the system's reliability [3]. Thus, to satisfy user requirements and avoid system threats, power consumption of modern mobile devices must be managed appropriately. To manage dynamic power consumption, modern mobile systems utilize OS-level approaches such as dynamic voltage/frequency scaling (DVFS) and dynamic power management (DPM) [4].

User scenarios on mobile devices have a wide range of operation characteristics (in this paper, operation characteristics of user scenarios are called *scenario characteristics*).

The associate editor coordinating the review of this manuscript and approving it for publication was Jing Bi.

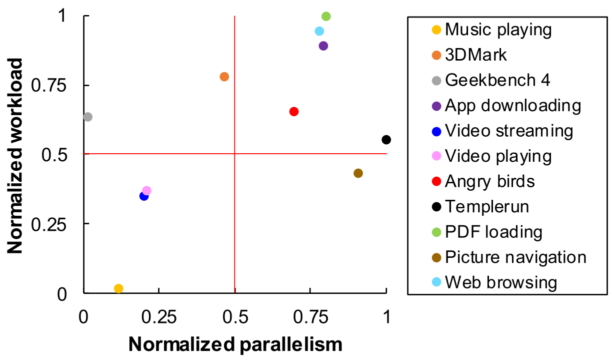


FIGURE 1. Normalized parallelism and workload processed by CPU for diverse user scenarios.

For example, user scenarios have various combinations of parallelism and workload (Fig. 1). In Fig. 1, parallelism is the number of runnable/running threads, and workload is the number of CPU clock cycles. Scenario characteristics directly affect the execution behavior of mobile systems. In this example, depending on the parallelism and workload, the number of active CPU cores and their usage will be determined. Thus, to provide effective power management regardless of the scenario that is running on the system, power management policies must consider the scenario characteristics.

Modern mobile devices are highly user-interactive, and scenario characteristics can change depending on user input. For example, during web browsing, user input determines the amount of information and data types that will be included in the loaded page, and scenario characteristics are different when different pages are loaded. However, future input is inherently impossible to predict in the mobile systems. Therefore, user-interactive mobile systems require runtime power management, and the power management policies of mobile systems must monitor scenario characteristics and adapt instantly to changes in characteristics.

In this paper, we propose a new power management policy that can increase the energy efficiency of modern mobile devices for various user scenarios. The proposed policy is designed as a DVFS policy for multicore CPUs, to control the operating frequencies of CPU clusters. To prevent energy waste and performance loss in advance, the proposed policy adjusts CPU frequencies proactively. To evaluate the suitability of the proposed method for modern mobile systems, we investigated whether the proposed policy provides satisfactory QoS, which is an important metric to evaluate the responsiveness of user-interactive mobile systems, and whether the policy balances QoS with consumed energy effectively. Our research makes three contributions:

1) The proposed policy performs DVFS for CPU clusters by considering thread-level parallelism and the requested amount of work; these characteristics can be used to represent the behavior of various user scenarios. Especially, the proposed policy uses thread-level parallelism of the requested work, so that it can consider the available hardware resources (in this research, the number of active cores) when it makes DVFS decisions; this ability is distinct from existing scenario-aware policies, which consider the amount of work only.
2) The proposed policy uses scenario characteristics that can be predicted during runtime; in this way our policy is unlike existing policies that require the preliminary knowledge of target scenarios. Thus, the proposed policy is appropriate for modern mobile devices in which online power management is essential.
3) To demonstrate the practicality of the proposed policy, we evaluated the policy by using real-world user scenarios that are frequently executed by end users. Besides, to demonstrate the effectiveness of the proposed policy for user-interactive mobile devices, we evaluated whether the proposed policy provides satisfactory QoS and maintains a good balance between energy efficiency and QoS.

The rest of this paper is organized as follows. Section II reviews previous research for effective CPU DVFS and presents the motivation of our research. Section III describes scenario characteristics used in the proposed policy. Section IV describes the proposed scenario-aware CPU DVFS policy. Section V presents experimental results to verify the effectiveness of the proposed policy. Section VI concludes the paper.

II. RELATED WORK

DVFS is one of the widely used techniques to manage power consumption; it scales voltage and frequency settings of the target component. Dynamic power is linearly proportional to the square of the voltage and to operating frequency [5]. Suitable voltage and frequency scaling lets the target component consume just enough dynamic power while satisfying the requested QoS. Furthermore, consumption of a reasonable level of dynamic power can prevent excessive increase in on-chip temperature. Therefore, an appropriate DVFS algorithm can contribute both to reducing dynamic power consumption and to preventing loss of system reliability due to various thermal problems such as hot spots and overheating.

In this research, we focus on DVFS policies for multicore CPUs. Commercial mobile devices with Linux-based OSes (e.g., Android) contain predefined DVFS policies (*CPUFreq* governors) for CPUs; examples include *ondemand*, *interactive*, and *performance* [4], [6]. However, CPUFreq governors do not adapt to scenario characteristics. To overcome the limitation of CPUFreq governors in the commercial devices, approaches in [7]–[18] attempt to adjust CPU frequencies in response to scenario characteristics. However, these approaches are not appropriate for general, user-interactive mobile systems, for two reasons. First, approaches in [7]–[14] focus on the limited types of user scenarios, and therefore are ineffective for the user scenarios beyond the target types. Second, approaches in [15]–[18] require preliminary knowledge of target scenarios, and therefore cannot be used for modern mobile devices in which online power management is essential.

Approaches in [7]–[14] consider scenario characteristics of the specific types of user scenarios; these approaches are impractical, or even inapplicable, when the running scenarios are not among the target types. Approaches in [7]–[9] focus on power management for gaming scenarios. These approaches go through an offline phase that uses a limited set of gaming scenarios to build performance and power models by observing frames-per-second (FPS) and power behavior when utilization and frequency of target components change. Then, these approaches go through an online phase to scale CPU operating frequencies by using the pre-constructed performance and power models in a way to increase estimated energy efficiency. However, these approaches construct performance and power models only from a limited set of gaming scenarios, and are therefore only useful within these and similar gaming scenarios.

Some other approaches [10]–[12] are applicable to a wider range of multimedia-centric user scenarios. One approach [10] uses machine learning to construct performance and power models by considering a training set composed of multimedia-centric user scenarios with diverse scenario characteristics. To optimize energy efficiency for multimedia-centric scenarios, the approach in [11] exploits the frame-level time slack and the approach in [12] uses the fluctuation in FPS. These approaches consider characteristics of a wider range of scenarios than the approaches described in [7]–[9], but are not suitable for user scenarios in which FPS is not a vital factor. Two approaches in [13], [14] attempt to optimize energy efficiency for web browsing, but they exploit the characteristics of web browsing such as HTML tags and DOM tree nodes, and therefore cannot be used for other types of user scenarios.

Some methods [15]–[18] were developed to increase energy efficiency for diverse types of user scenarios. However, these approaches require preliminary information about the user scenarios that will be executed; these approaches cannot be used for runtime power management. The approach in [15] classifies user scenarios into three categories (performance-sensitive, QoS-sensitive, and insignificant) before execution, and applies different DVFS strategies depending on the target scenario's categories. Some other approaches [16], [17] require target performance (e.g., target FPS and throughput) of user scenarios to calculate the optimal CPU frequency. Another approach in [18] goes through an offline phase to profile the expected speedup and energy for various CPU frequency levels under the execution of the target scenario, and uses the profiled data to guide online power management.

The existing scenario-aware DVFS policies cannot cover a wide range of user scenarios [7]–[14] and are not suitable for online power management of mobile devices [15]–[18]. To overcome these drawbacks of the existing approaches, we propose a new scenario-aware DVFS policy. The proposed policy considers scenario characteristics that can represent the behavior of a wide range of scenarios. Thus, it can adapt to diverse user scenarios. Furthermore, the proposed policy does not require preliminary knowledge of target scenarios, and therefore can be used for online power management of modern mobile devices.

III. PRELIMINARY WORK

In this section, we define the scenario characteristics that the proposed policy considers, and describe how we can obtain these characteristics online. For effective management of power consumption regardless of user scenarios, scenario-aware policies must use appropriate characteristics, which can represent a wide range of execution behavior of mobile systems for diverse user scenarios. Furthermore, although several existing scenario-aware policies use performance monitoring counters (PMCs) to consider the characteristics during runtime [18], [19], the task of accessing and reading PMCs is challenging or impossible for many mobile platforms [20]. Therefore, to define scenario characteristics, we also consider whether runtime access to the characteristic is achievable for diverse mobile platforms.

Most existing scenario-aware DVFS policies use the scenario characteristics that indicate the amount of work, such as FPS, the number of clock cycles, and CPU usage. However, considering the amount of work only is not sufficient to enable estimation of the processing speed required to process the requested work by using available resources. In fact, most user scenarios consist of multiple threads. When a user scenario is running, the number of concurrently-runnable threads can change depending on dependency among threads and their CPU affinity. If the number of concurrently-runnable threads is smaller than the core count of the target platform, the number of cores that may engage in the execution is also smaller than the original core count. Therefore, even though the operating frequency is the same, the amount of work that the CPU can process can be different depending on the thread-level parallelism. Thus, the proposed policy considers both the amount of work and the thread-level parallelism to provide the best operating frequency for the requested amount of work and the number of utilizable cores.

In this research, we use two scenario characteristics to adjust CPU frequencies: (1) thread-level parallelism (TLP) [21] to consider parallelism, and (2) clock cycle count per cluster ($c_{cluster}$) to consider the amount of work. Both characteristics are calculated from the utilization of CPU cores.[1]

A. THREAD-LEVEL PARALLELISM

1) DEFINITION

TLP [21] indicates parallelism of user scenarios running on the target mobile devices when one or more CPU cores are active. The core count of the target mobile devices and data

[1]In mobile systems with Linux-based OSes, each CPU core's utilization can be calculated using the accumulated active and inactive times of each core, which are provided by using a *jiffy* unit. One jiffy means a period of interrupts triggered by the kernel timer system. The default jiffy interval is set as 10 ms [22].

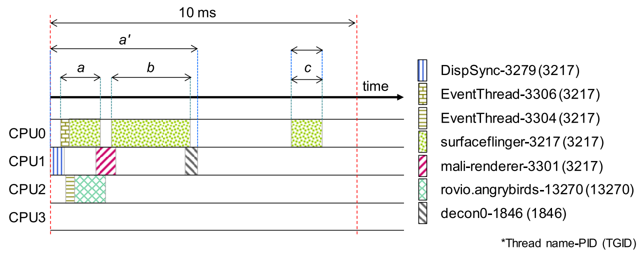


FIGURE 2. Context-switching events captured on Galaxy S7 when Angry Birds Classic was executed; each color bar indicates execution of the corresponding thread.

locality of threads are reflected in the TLP.

$$TLP_i = \frac{\sum_{k=1}^{N_i} tf_{k,i}k}{1 - tf_{0,i}} \tag{1}$$

The proposed policy uses TLP of each CPU cluster. Thus, a variable i is added as an index of the CPU clusters. TLP of CPU cluster i (Eq. 1) is calculated by dividing the sum of fractions of active times of all cores in the CPU cluster by the fraction of active time of the CPU cluster. In Eq. 1, $tf_{k,i}$ is a fraction of time that k cores in the cluster i are concurrently active (a fraction of active time), $tf_{0,i}$ is a fraction of idle time of the cluster i, $1 - tf_{0,i}$ is a fraction of active time of the cluster i, and N_i is the core count of the CPU cluster i.

2) ESTIMATION OF THREAD-LEVEL PARALLELISM

To achieve the exact value of $1-tf_{0,i}$ in Eq. 1, we must track when context-switching events occur, and on what CPU cores they occur. Then, from the tracked information, we must extract the amount of time that threads occupy each core. Performing this process online is challenging or impossible for many mobile platforms due to the limited size of the trace buffer. Thus, instead of calculating the exact value of $1-tf_{0,i}$ online, the proposed policy estimates $1-tf_{0,i}$ by considering utilization values of CPU cores.

When representing utilization from 0 (idle) to 1 (fully active), the lower bound of $1 - tf_{0,i}$ is the utilization of CPU cluster i when active times of CPU cores overlap as much as possible. On the contrary, the upper bound of $1 - tf_{0,i}$ is the utilization of CPU cluster i when active times of CPU cores have minimum overlaps.

The frequency level selected by the proposed policy tends to increase as TLP decreases (Section IV-B). This relationship means that if we estimate $1 - tf_{0,i}$ as the upper bound, the estimation error can cause significant energy waste. Thus, the proposed policy estimates $1 - tf_{0,i}$ as the lower bound; under the assumption that active time of the busiest core covers active times of the other cores, $1 - tf_{0,i}$ can be calculated as the utilization of the busiest core in cluster i during one sampling period. Active time of a CPU core is the total time occupied by all threads running on the core, so the estimated $1 - tf_{0,i}$ reflects the execution behavior of both normal and real-time threads. To clarify estimation technique, Fig. 2 provides an example of context-switching events captured on the LITTLE cluster (CPU0-3) of Galaxy S7. During the 10 ms, the actual $1 - tf_{0,i}$ is $(a' + c)$ divided by 10 ms and the estimated $1 - tf_{0,i}$ is $(a+b+c)$ divided by 10 ms.

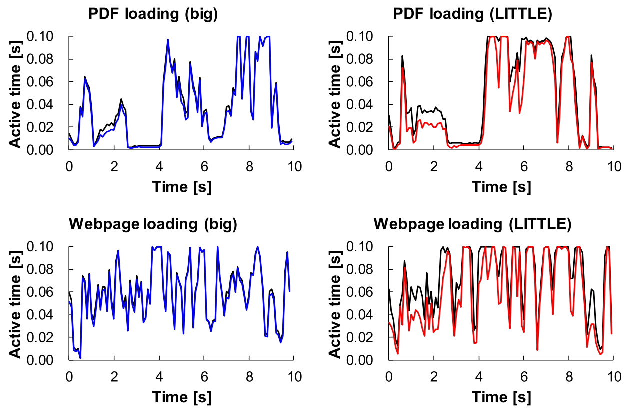


FIGURE 3. Examples of measured (black) and estimated (blue and red) active times of big and LITTLE CPU clusters.

To the best of our knowledge, this is the first attempt to trace TLP online. As shown in Fig. 3, the tendencies of the measured and estimated active times are generally consistent, although subtle inconsistency exists depending on scenarios. We discuss effectiveness of the current estimation technique in Section V-C-3.

B. CLOCK CYCLE COUNT PER CLUSTER

1) DEFINITION

In this paper, $c_{cluster}$ indicates the number of effective clock cycles that the cores in the corresponding CPU cluster need to consume. The number of CPU clock cycles is generally proportional to the amount of work allocated to the CPU [16], [23], so we define $c_{cluster}$ to represent the amount of work that each CPU cluster must process.

The number of clock cycles consumed by a core during a specific period can be calculated by integrating the multiplication of core utilization and frequency over the period. However, when a core's utilization is 1, there is a possibility that the core is a bottleneck due to the low operating frequency; in this case, $c_{cluster}$ can be calculated lower than the value actually requested. Thus, this paper introduces an up-scaling coefficient, which scales up $c_{cluster}$ when one or more cores in the cluster are fully active during the given period. Eq. 2 defines the number of clock cycles consumed by the j^{th} core in cluster i, where $u_{j,i}$ is the utilization of the j^{th} core in the cluster i, f_i is the operating frequency of cluster i, t_p is a duration of one sampling period, and s_i is an up-scaling coefficient of cluster i:

$$c_{j,i} = \begin{cases} t_p u_{j,i} f_i, & u_{j,i} < 1 \\ t_p s_i u_{j,i} f_i, & u_{j,i} = 1 \end{cases} \tag{2}$$

TABLE 1. Given parameters for up-scaling coefficient determination.

Param.	Description	Range
n_{fa}	The number of fully-active cores	$1 \sim N_i/2$ [a]
f_{prev}	Previous frequency level	Available frequency levels [b]
u_t	Total utilization of cores	$n_{fa} \sim \{n_{fa} + 0.99\cdot(N_i - n_{fa})\}$ [c]

[a]In real life, rarely more than half of CPU cores become bottlenecks at the same time.
[b]In mobile systems with Linux-based OSes, available frequency levels can be generally checked from the *scaling_available_frequencies* file.
[c]Investigation is performed by increasing u_t by 0.01 step.

Eq. 3 defines $c_{cluster}$ of cluster i, which is the sum of $c_{j,i}$ for all cores in cluster i:

$$c_{cluster,i} = \sum_{j=0}^{N_i-1} c_{j,i} \tag{3}$$

2) UP-SCALING COEFFICIENT DETERMINATION

When one or are bottlenecks, $c_{cluster}$ can be underestimated. To avoid this problem, we introduce an up-scaling coefficient. Up-scaling coefficients differ depending on platforms and cpu clusters. Table 1 explains parameters (n_{fa}, f_{prev}, and u_t) used for the coefficient determination. Ranges of n_{fa}, f_{prev}, and u_t are dependent on platforms and cpu clusters. n_{fa} is the number of fully-active cores. f_{prev} is the previous frequency level. u_t is the total utilization of cores in the target cpu cluster when n_{fa} cores are fully active. Effectiveness of coefficient candidates needs to be evaluated for given ranges of n_{fa}, f_{prev}, and u_t.

The up-scaling coefficient is determined considering two aspects: relaxation of the CPU bottleneck, and minimal energy increase. First, we need to investigate whether each coefficient candidate contributes to the effective relaxation of the cpu bottleneck. When one or more cores are bottlenecks, the bottleneck phenomenon can be alleviated by increasing the operating frequency. Thus, a specific coefficient candidate s is considered to contribute to the bottleneck relaxation if the next frequency selected under the corresponding candidate $f_{next}(s_i = s, u_t)$ is higher than $f_{next}(s_i = 1, u_t)$. Second, we need to investigate how much dynamic power increases due to each coefficient candidate. In general, operating frequency and voltage are linearly correlated [24], [25]. Dynamic power consumption is proportional to the square of voltage and to frequency, so it can be modeled as a function of the cube of frequency [25], [26]. Therefore, the ratio of dynamic power increment caused by a coefficient candidate s is calculated as the cube of $f_{next}(s_i = s, u_t)$ divided by the cube of $f_{next}(s_i = 1, u_t)$.

The up-scaling coefficient s_i of CPU cluster i is determined by evaluating the effectiveness of coefficient candidates. For the numerical evaluation, we define R_{perf}, R_{power}, and R_{eff} (Table 2). R_{perf} represents the possibility of CPU bottleneck relaxation, i.e., the proportion of cases in which the $f_{next}(s_i = 1, u_t)$ and $f_{next}(s_i > 1, u_t)$ differ by one level. R_{power} is the ratio of dynamic power increment due to coefficient candidates. R_{eff} represents the effectiveness of

TABLE 2. Parameters to evaluate coefficient candidates.

Param.	Description	Calculation
R_{perf}	Possibility of CPU bottleneck relaxation	Proportion of cases when $f_{next}(s_i{=}1, u_t)$ and $f_{next}(s_i{>}1, u_t)$ differ by one level [a]
R_{power}	Ratio of dynamic power increment resulted by s_i	Average of $(f_{next}(s_i{>}1, u_t)/f_{next}(s_i{=}1, u_t))^3$ for all u_t points
R_{eff}	Numerical representation of the overall effectiveness; criterion for final determination	R_{perf}/R_{power}

[a]To avoid excessive increase in energy consumption, we only count the case that $f_{next}(s_i{>}1, u_t)$ and $f_{next}(s_i{=}1, u_t)$ differ by one level.

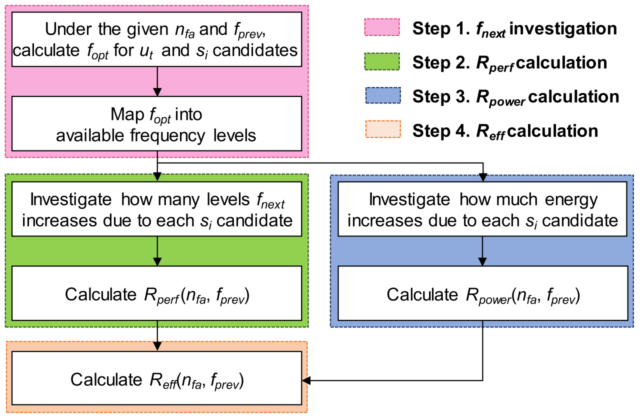

Under the given n_{fa} and f_{prev}, calculate f_{opt} for u_t and s_i candidates
Map f_{opt} into available frequency levels
Investigate how many levels f_{next} increases due to each s_i candidate
Calculate $R_{perf}(n_{fa}, f_{prev})$
Investigate how much energy increases due to each s_i candidate
Calculate $R_{power}(n_{fa}, f_{prev})$
Calculate $R_{eff}(n_{fa}, f_{prev})$
Step 1. f_{next} investigation
Step 2. R_{perf} calculation
Step 3. R_{power} calculation
Step 4. R_{eff} calculation

FIGURE 4. Effectiveness evaluation of a coefficient candidate for specific nfa and fprev; NF is the number of available frequency levels.

coefficient candidates considering both bottleneck relaxation and minimal energy increase, and is used as the criterion for final determination of the up-scaling coefficient.

The effectiveness evaluation is performed by quantifying R_{perf}, R_{power}, and R_{eff} under the given n_{fa}, f_{prev}, and u_t ranges. Fig. 4 displays the procedure of effectiveness evaluation of each coefficient candidate for given n_{fa} and f_{prev} values and Fig. 5 shows examples of the steps described in Fig. 4. First, in step 1, the profile of $f_{opt}(s_i, u_t)$ is investigated (Section IV-B), and $f_{opt}(s_i, u_t)$ is mapped to the closest level among available frequency levels; this process generates the $f_{next}(s_i, u_t)$ profile. in Fig. 5a, $f_{next}(s_i, u_t)$ is generated by quantizing $f_{opt}(s_i, u_t)$ to 450, 575, 700, 775, or 850 MHz. Then, using the obtained $f_{next}(s_i, u_t)$ profile, $R_{perf}(\mathrm{n}_{fa}, f_{prev})$ and $R_{power}(n_{fa}, f_{prev})$ are calculated in steps 2 and 3. In Fig. 5b, for each coefficient candidate, the frequency level difference between $f_{next}(s_i = 1, u_t)$ and $f_{next}(s_i > 1, u_t)$ is determined, then used to calculate $R_{perf}(n_{fa}, f_{prev})$. in Fig. 5b, when $f_{next}(s_i = 1, u_t)$ is 450 MHz, the frequency level difference is one, two, three, and four if $f_{next}(s_i > 1, u_t)$ is 575 MHz, 700 MHz, 775 MHz, and 850 MHz, respectively. in Fig. 5c, for each coefficient candidate, the ratio of dynamic power increment is calculated, and $R_{power}(n_{fa}, f_{prev})$ is calculated by averaging the ratios of dynamic power increment of all u_t points. Finally, in step 4, $R_{eff}(n_{fa}, f_{prev})$ of each coefficient candidate is calculated.

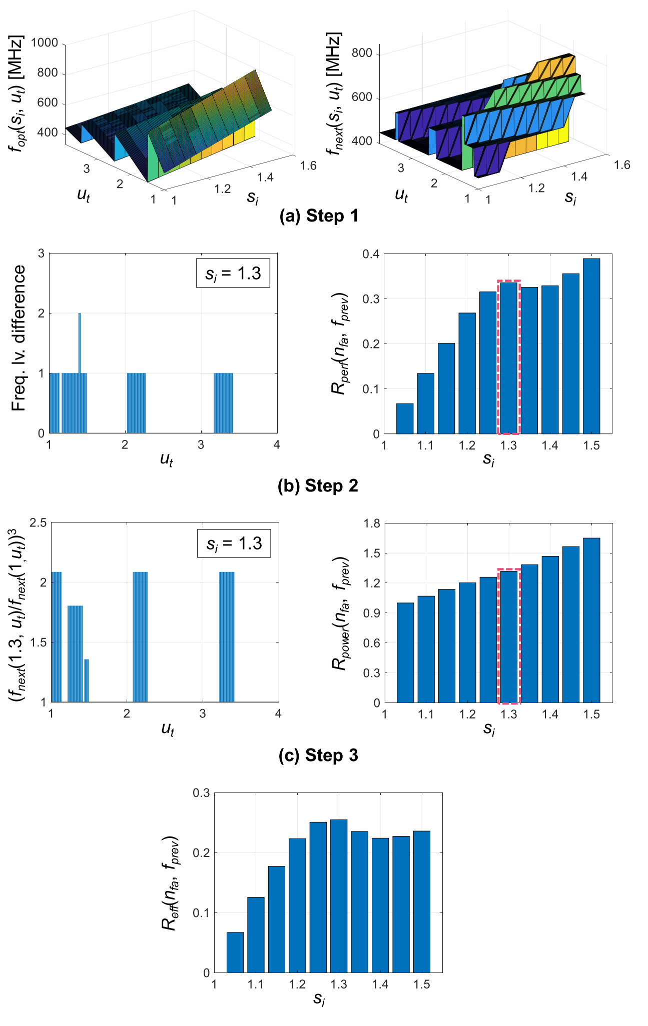


FIGURE 5. Example of the effectiveness evaluation when $N_i = 4$, $n_{fa} = 1$, and $f_{prev} = 450$ MHz; available frequency levels are 450, 575, 700, 775, and 850 MHz.

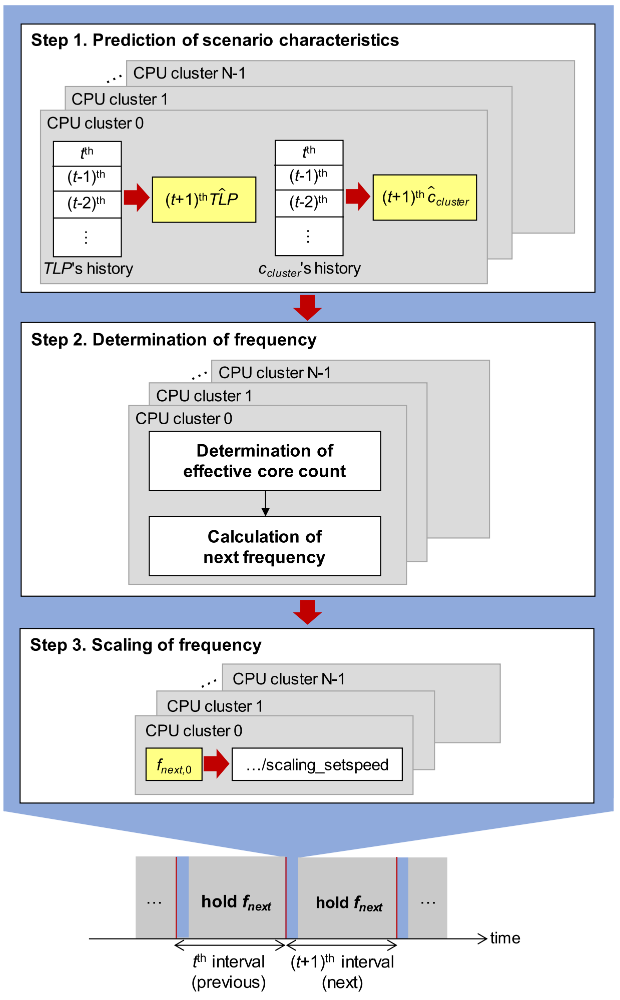

Step 1. Prediction of scenario characteristics
CPU cluster N-1
CPU cluster 1
CPU cluster 0
TLP's history
$c_{cluster}$'s history
Step 2. Determination of frequency
Determination of effective core count
Calculation of next frequency
Step 3. Scaling of frequency
.../scaling_setspeed
hold f_{next}
time
tth interval (previous)
$(t+1)$th interval (next)

FIGURE 6. Flowchart of the proposed scenario-aware CPU DVFS policy.

Fig. 5d shows $R_{eff}(n_{fa}, f_{prev})$ obtained by dividing $R_{perf}(n_{fa}, f_{prev})$ by $R_{power}(n_{fa}, f_{prev})$ for each coefficient candidate.

Steps 1-4 in Fig. 4 are iterated by traversing given n_{fa} and f_{prev} ranges. After all iterations, the average value of $r_{eff}(n_{fa}, f_{prev})$ is calculated for all available frequency levels as

$$R_{eff.avg}(n_{fa}) = \sum\nolimits_{m=0}^{N_F-1} R_{eff}(n_{fa}, f_{available}(m))/N_F \quad (4)$$

where $f_{available}(m)$ is the m^{th} available frequency level and n_f is the number of available frequency levels. Finally, s_i is determined as the coefficient candidate that maximizes the average $r_{eff.avg}(n_{fa})$.

IV. PROPOSED SCENARIO-AWARE POLICY

The proposed policy performs scenario-adaptive power management by adjusting CPU frequencies. Differentiated from the existing approaches in [7]–[14], the proposed policy is adaptive to various user scenarios because it uses scenario characteristics that can represent the execution behavior of a wide range of scenarios; TLP and $c_{cluster}$ described in Section III are used to consider parallelism and the amount of requested work. Besides, the proposed policy is suitable for online power management because it does not require preliminary knowledge of target scenarios, whereas the approaches in [15]–[18] do require it. The proposed policy targets the platform, which defines the DVFS domain as a CPU cluster, and supports both asymmetric and symmetric multicore CPUs. The proposed policy manages power consumption of each CPU cluster independently, so the execution time of the proposed policy increases linearly as the number of CPU clusters increases. Power management quality of the proposed policy is not affected by the size of data used for the scenario's execution; the target platform's performance can be sensitive to the data size depending on the platform's hardware specification (e.g., cache size).

The proposed policy (Fig. 6) consists of three steps: (1) prediction of scenario characteristics, (2) determination of frequency, and (3) scaling of frequency. The proposed policy performs power management periodically by conducting these three steps at the beginning of the next interval, and holds the determined frequency until the end of the next interval.

In the first step, the proposed policy predicts scenario characteristics described in Section III. In the second step, the proposed policy uses the predicted characteristics to calculate the optimal next frequency. In the last step, the proposed policy scales the CPU frequency by using the *userspace* governor [4], [6] pre-defined in Linux-based OSes, which enables frequency adjustment at the user level. The following sections focus on the first and second steps.

A. PREDICTION OF SCENARIO CHARACTERISTICS

For proactive power management, the proposed policy predicts the scenario characteristics of the next interval by using the history values of *TLP* and $c_{cluster}$ that have been sampled during previous intervals. In this paper, $T\hat{L}P_i$ and $\hat{c}_{cluster,i}$ indicate predicted TLP_i and $c_{cluster,i}$ of the next interval.

To improve the prediction accuracy compared to existing proactive approaches, we combine a normalized least-mean-square (NLMS) linear predictor [27] with an exponentially-weighted moving average (EWMA) predictor, which is generally used by existing proactive policies [16], [28].

The NLMS linear predictor is appropriate to identify parameters of dynamic systems such as mobile devices [27]. Eq. 5 is a basic form of a linear predictor,

$$\hat{x}[t+1] = \sum_{k=0}^{N-1} w_k[t]x[t-k] = \bar{W}[t]^T\bar{X}[t] \qquad (5)$$

where $x[t]$ is the target parameter x at the t^{th} interval, $\hat{x}[t+1]$ is the predicted $(t+1)^{\text{th}}x$, N is the order of the predictor, and $w_k[t]$, for $k = 0, \ldots, N-1$, are a linear predictor's weights at the t^{th} interval, which are generally initialized to zero in an NLMS linear predictor. The weights of the NLMS linear predictor are calculated as

$$e[t] = x[t] - \hat{x}[t] \qquad (6)$$

$$\bar{W}[t] = \bar{W}[t-1] + \frac{\mu e[t]}{\left\|\bar{X}[t-1]\right\|^2}\bar{X}[t-1] \qquad (7)$$

where $e[t]$ is the t^{th} prediction error and μ is the learning rate. The NLMS linear predictor recursively updates its weights (Eqs. 6-7) in a direction to reduce the prediction error. One critical drawback of this predictor is that the prediction error can be very large if the weights diverge [28].

To overcome this limitation, when the prediction error exceeds a pre-defined threshold, the proposed policy resets the weights by using a widely used form for EWMA predictor's weights (Eqs. 8-9).

$$\lambda = \frac{2}{N+1}, \qquad (8)$$

$$c_k[t] = \lambda(1-\lambda)^k. \qquad (9)$$

We set the predictor with empirically obtained parameters.[2]

TLP prediction examples (Fig. 7) were obtained by applying the NLMS linear predictor, the EWMA predictor, and the combination of NLMS linear and EWMA predictors when a user used Chrome (a web browser) for 60 seconds.

[2] We set N as 8, μ as 0.074, and the threshold of the prediction error as 70 %.

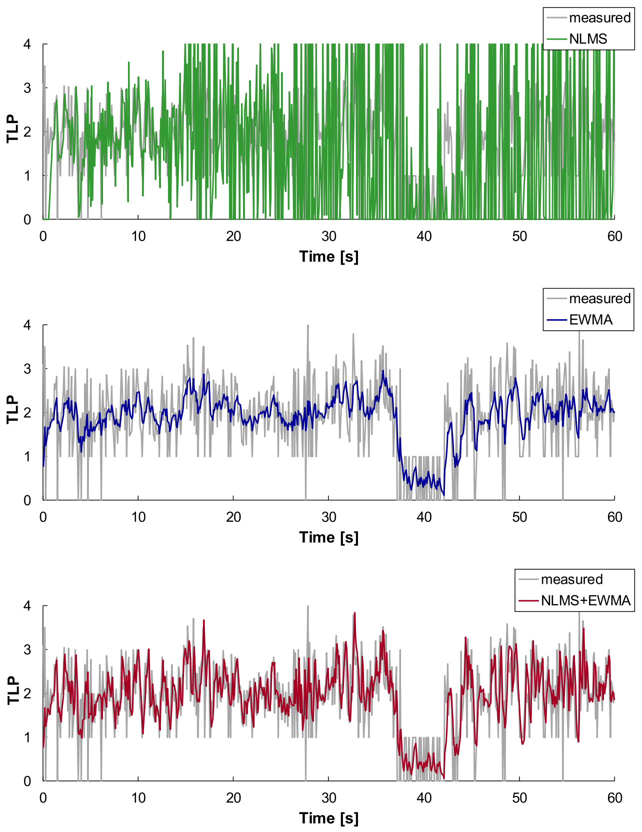


FIGURE 7. Prediction examples obtained using the NLMS linear predictor, the EWMA predictor, and the combination of NLMS linear and EWMA predictors.

The NLMS linear predictor followed peak values with high accuracy initially, but after 15 seconds, the prediction error became extremely large because the weights diverged. The EWMA predictor achieved relatively low prediction error, but could not sharply follow peak values. The combination of NLMS linear and EWMA predictors overcame both of these limitations.

B. DETERMINATION OF FREQUENCY

In this step, the proposed policy uses $T\hat{L}P$ and $\hat{c}_{cluster}$ to determine the next operating frequencies of the CPU clusters. This step consists of two phases: (1) determination of effective core count and (2) calculation of next frequency.

1) DETERMINATION OF EFFECTIVE CORE COUNT

In this paper, *effective core count* means the number of cores that are utilizable for the scenario's execution. The proposed policy calculates the next effective core count of each cluster by using $T\hat{L}P$, which indicates the number of threads that are expected to be concurrently-runnable at the next interval. The next effective core count $n_{eff,i}$ of CPU cluster i is calculated by rounding $T\hat{L}P_i$ as

$$n_{eff,i} = \begin{cases} \left\lfloor T\hat{L}P_i + 0.5 \right\rfloor, & T\hat{L}P_i \geq 0.5 \\ 1, & otherwise. \end{cases} \qquad (10)$$

When $T\hat{L}P_i$ is lower than 0.5 in the next interval, one core will be effective to process the work requested to the CPU cluster i. Thus, when $T\hat{L}P_i$ is lower than 0.5, we set $n_{eff,i}$ as 1 instead of 0, which is the rounding result of a value lower than 0.5.

2) CALCULATION OF NEXT FREQUENCY

The proposed policy determines the next frequency as the frequency level that is the most appropriate to process the requested amount of work by using the utilizable CPU cores. The key strategy of the proposed policy is to provide 'just enough' CPU processing speed during execution of various scenarios.

In this step, the proposed policy calculates the optimal level of the next frequency of each CPU by considering the strategy described above. Then, the proposed policy selects the next frequency among the available frequency levels.

First, the proposed policy calculates the optimal frequency. Eq. 11 describes the optimal frequency $f_{opt,i}$ of CPU cluster i. $\hat{c}_{cluster,i}$ indicates the clock cycle count that is expected to be requested, and $n_{eff,i}$ is the expected number of effective cores. Thus, $f_{opt,i}$ is the frequency level that provides 'just enough' processing speed to process $\hat{c}_{cluster,i}$ using $n_{eff,i}$ cores at the next interval.

$$f_{opt,i} = \frac{\hat{c}_{cluster,i}}{n_{eff,i} t_p} \tag{11}$$

After calculating $f_{opt,i}$, the proposed policy sets the next frequency $f_{next,i}$ of CPU cluster i as the closest level to $f_{opt,i}$ among available frequency levels that are pre-defined in the target platform.

V. EXPERIMENTAL STUDY

We conducted an experimental study to demonstrate the effectiveness of the proposed scenario-aware policy. The following paragraphs describe the details of the experimental setup and results.

A. EXPERIMENTAL SETUP

1) TARGET PLATFORM

The target platform is V2M-Juno development platform [29] with Android OS. The target platform contains CPU based on big.LITTLE architecture, where Cortex-A57 dual cores (big CPU cluster) are paired with Cortex-A53 quad cores (LITTLE CPU cluster). Each CPU cluster has five available frequency levels: for the big CPU cluster they are 450, 625, 800, 950, and 1,100 MHz; for the LITTLE CPU cluster they are 450, 575, 700, 775, and 850 MHz.

2) EVALUATION METRICS

General user scenarios that run on mobile systems interact directly with end users. Thus, when evaluating performance of mobile systems, QoS is much more important than raw processing speed.

Mobile systems fail to guarantee satisfactory QoS in two major ways [30]: they provide performance that is less than the human-tolerable limit P_U or they provide performance that exceeds end users' expectations P_I. Violation of P_U causes QoS violations, and end users may abandon to use the systems. Violation of P_I makes the systems use unnecessary energy, energy efficiency and battery life are reduced, thermal problems occur, and the system's reliability deteriorates. Thus, evaluation of QoS must consider whether performance is higher than P_U and lower than P_I.

As the primary goal of the proposed policy is to improve the energy efficiency of mobile systems, we evaluated energy and QoS of the proposed policy. To enable numerical evaluation of how the proposed policy manages the trade-off between energy consumption and QoS, this paper introduces a metric *energy per QoS* (*EPQ*) in Eq. 12. Lower EPQ means that the policy balances the trade-off between energy and QoS more effectively. In the study, *performance* in Eq. 12 indicates FPS and the inverse of event latency

$$EPQ = \frac{energy}{performance} \tag{12}$$

This paragraph describes how we measured CPU clusters' energy consumption, and performance. We measured CPU clusters' energy consumption using energy meter registers adopted in V2M-Juno development platform. The energy meter registers in the target platform provide cumulative energy consumption of each CPU cluster from the booting. To obtain the energy consumed during scenario's execution, we measured the cumulative energy consumption at starting and ending points of the execution, and then calculated the difference between the two measured values. We also measured event latency by recording the visual output of user scenarios displayed on the connected monitor, and by analyzing the recorded videos using *PotPlayer* [31], a commercial video player that provides 1-ms time resolution. We measured FPS by using *dumpsys* [32], a tool that runs on Android OS by default; it provides information about system services.

3) TEST SCENARIOS

To demonstrate the practicality of the proposed policy, we conducted an experimental study when real-world user scenarios (Table 3) were executed. Zhu *et al.* have classified user scenarios by what metric must be used for QoS evaluation, and have provided the evaluation guideline (P_I and P_U) of each scenario category [30].

The first category is composed of "job delay" scenarios, which require minimal user input to trigger intensive jobs; in this category, P_I is 1 second and P_U is 10 seconds. The second category is composed of "response latency" scenarios, which consist of relatively a few events and have low event latency; in the case of web-related scenarios in this category, P_I is 1 second and P_U is 3 seconds. The last category contains "FPS" scenarios, which consist of many events and have low event latency, such as gaming and video playing; in this category, P_I is 60 FPS and P_U is 30 FPS.

We constructed and executed test scenarios by using RERAN [33], a macro program that records and replays user

TABLE 3. Basic test scenarios used for the experimental study.

QoS metric	Scenario	Description	QoS evaluation region	
			Starting point	Ending point
Job delay	AB-stld	In Angry Birds Classic, request the specific stage and play the game after loading.	Click the stage	Loading is completed
	PV-pdfld	Using PDF viewer, open the pdf file, and scroll the loaded file.	Double click the file	
Response latency	HV-pgld	Type 'amazon.com' at the address bar, press 'Enter' key, and scroll the loaded page; the page mainly consists of images (heavy page).	Press 'Enter'	
	LT-pgld	In www.google.com, type 'theorem' at the search box, press 'Enter' key, and scroll the loaded page; the page mainly consists of texts (light page).	Press 'Enter'	
	YT-vld	In Youtube, click the video and watch the loaded video.	Click the video	
FPS	YT-vstr	In Youtube, when the video is paused and full-screen mode is activated, click the 'play' button.	Play the video	Execution ends
	GL-vpl	In Gallery, click the 'play' button of the video.	Play the video	
	JJ-gpl	Play the game Jetpack Joyride.	Start execution	

QoS evaluation region indicates the interval where we measured QoS (job delay, response latency, and FPS).

events (e.g., typing, clicking, and scrolling). We constructed each basic test scenario to have duration of 15 seconds.

4) BENCHMARK POLICIES

We compared the proposed scenario-aware policy with two CPUFreq governors that are in the commercial devices [4], [6], and two existing approaches that have been proposed in the literature [12], [34]. The *interactive* governor [4], [6] (*int*) is widely used on modern Android devices; it adjusts CPU frequencies by considering the current CPU utilization. The *powersave* governor [4], [6] (*pws*) is another CPUFreq governor; it always uses the lowest CPU frequency without considering other factors such as CPU usage. The approach in [34] scales CPU frequencies in a way to maximize CPU utilization. We refer to this approach as the *power-minimizing* policy (*pwm*). The approach in [12] mainly focuses on FPS scenarios; it requires an offline phase to obtain initial values of the scenario-specific coefficients. Compared to the other policies focusing on FPS scenarios in Section II, this approach has the flexibility to a wider range of FPS scenarios and has a relatively simple offline phase. We refer to this approach as the *multimedia-centric* policy (*mmc*). For CPUFreq governors, we used parameter settings pre-defined in the kernel. For *pwm* and *mmc* policies, we followed parameter selections and suggestions described in [34] and [12], respectively.

B. PARAMETER DETERMINATION

1) UP-SCALING COEFFICIENT

In this paper, we introduce an up-scaling coefficient to avoid underestimation of $c_{cluster}$ due to possible CPU bottlenecks. To quantify the influence of the up-scaling coefficient on performance and energy consumption, we define R_{perf}, R_{power}, and R_{eff}. R_{perf} is the possibility of CPU bottleneck relaxation, R_{power} is the ratio of dynamic power increment due to the coefficient, and R_{eff} is the effectiveness of the coefficient considering both bottleneck relaxation and the minimal increase

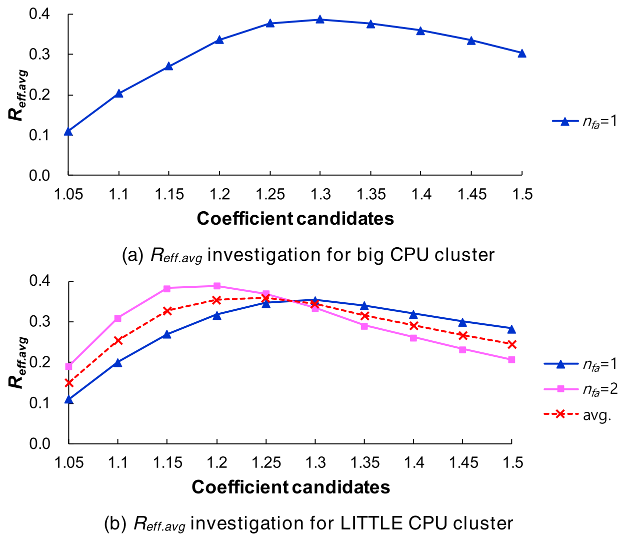


FIGURE 8. Reff.avg of big and LITTLE CPU clusters for ten coefficient candidates.

in energy consumption. Details of the investigation of R_{perf}, R_{power}, and R_{eff} are given in Section III-B-2.

This section describes how we determined the up-scaling coefficients s_{big} for big CPU cluster and s_{LITTLE} for LITTLE CPU cluster in the target board. To determine s_{big} and s_{LITTLE} for the target platform, $R_{eff.avg}$ of the big and LITTLE CPU clusters were calculated for ten coefficient candidates (Fig. 8). We investigated $R_{perf}(n_{fa}, f_{prev})$, $R_{power}(n_{fa}, f_{prev})$, and $R_{eff}(n_{fa}, f_{prev})$ for all available n_{fa} and f_{prev} values, and calculated the average R_{eff} for all candidates of f_{prev} $R_{eff.avg}$. A large $R_{eff.avg}$ means that the corresponding up-scaling coefficient effectively relaxes CPU bottlenecks while causing a small increase in energy consumption. The big CPU cluster of the target platform contains two cores, so we only considered when n_{fa} is 1 for the big CPU cluster.

In Fig. 8a, $R_{eff.avg}$ was highest when s_{big} is 1.30, and in Fig. 8b, the average $R_{eff.avg}$ was highest when s_{LITTLE} is

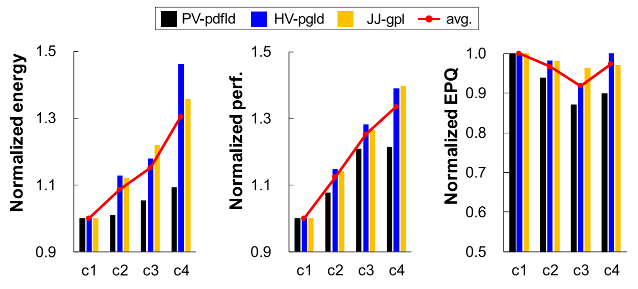


FIGURE 9. Normalized energy consumption, performance, and EPQ for four combinations of up-scaling coefficients (x-axis; Table 4); energy, performance, and EPQ were normalized to the values with c1.

TABLE 4. Combinations of up-scaling coefficients.

Up-scaling coefficient	c1	c2	c3[a]	c4
big (s_{big})	1.00	1.15	1.30	1.45
LITTLE (s_{LITTLE})	1.00	1.10	1.25	1.40

[a]c3 is the combination of up-scaling coefficients with the highest $R_{eff.avg}$.

1.25. To validate s_{big} and s_{LITTLE} which provided the highest $R_{eff.avg}$ for each CPU cluster, we quantified energy consumption, performance, and EPQ by using PV-pdfld, HV-pgld, and JJ-gpl, which process the large amount of work compared to the other scenarios in the same category (Table 3). We validated the coefficients (c3) with the highest $R_{eff.avg}$ values by comparing with three other combinations (Table 4): *no-scaling* (c1), *small coefficients* (c2), and *large coefficients* (c4). Consumed energy and performance tended to increase as the up-scaling coefficients were increased (Fig. 9). Compared to the others, the combination with the highest $R_{eff.avg}$ provided the lowest EPQ by balancing energy consumption and performance the most effectively. EPQ was highest when s_{big} and s_{LITTLE} were set as the no-scaling combination; the policy with the no-scaling combination could not relax CPU bottlenecks effectively. Combinations of small and large coefficients provided similar EPQs; the small coefficients provided lower performance, and the large coefficients provided lower energy efficiency. Considering the above observations (Fig. 8-9), we set s_{big} and s_{LITTLE} as 1.30 and 1.25.

2) PERIOD OF FREQUENCY ADJUSTMENT

The period of the proposed policy affects energy consumption and performance. In Linux-based OSes, CPU utilization is calculated using active and inactive times that are represented in 10-ms units [22]. If the period is too short, the resolution of CPU utilization can be too coarse. Scenario characteristics are obtained from CPU utilization, so the too-coarse resolution of CPU utilization degrades the accuracy of scenario characteristic prediction, so that the quality of online power management can also decrease. Furthermore, the too-short period increases the additional CPU utilization caused by the proposed policy (CPU runtime overhead) [35]; the excessive CPU runtime overhead causes unnecessary energy consumption by the cores [36]. In contrast, if the period is too long, the policy can lose responsiveness. The policy with low responsiveness cannot rapidly adapt to changes in scenario characteristics; this tendency can degrade QoS provided by the policy.

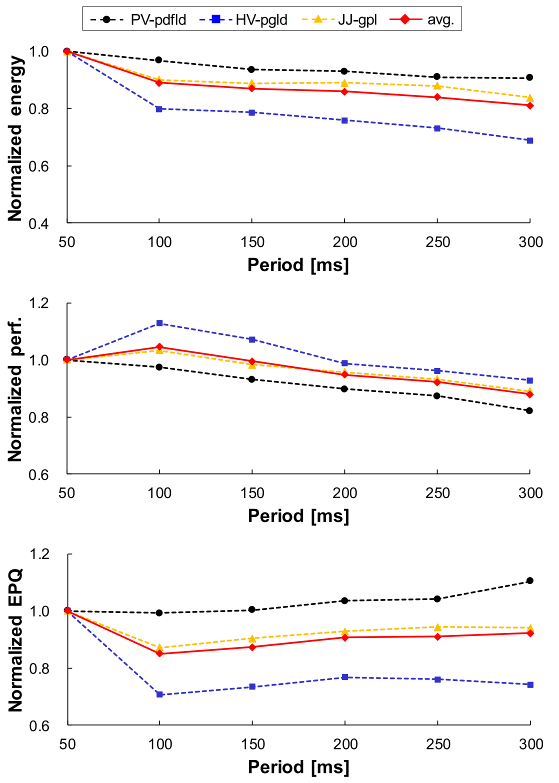


FIGURE 10. Normalized energy, performance, and EPQ for period candidates; energy, performance, and EPQ were normalized based on the values when the period is 50 ms.

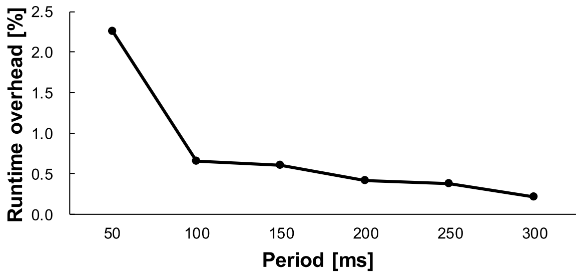


FIGURE 11. Average CPU runtime overhead depending on the period.

To determine the period, we quantified how the period affects energy consumption, performance, and EPQ, by using PV-pdfld, HV-pgld, and JJ-gpl. Fig. 10 shows energy, performance, and EPQ for various period candidates. Several observations were obtained. First, as the period decreases, energy consumption of CPUs tends to increase. Too-frequent execution of the policy burdened CPUs, so they wasted energy. The average CPU runtime overhead decreased as period increased (Fig. 11); the runtime overhead was about 3.4 times higher at a period of 50 ms than at a period of 100 ms. Because the proposed policy performs file reading to sample CPU cores'

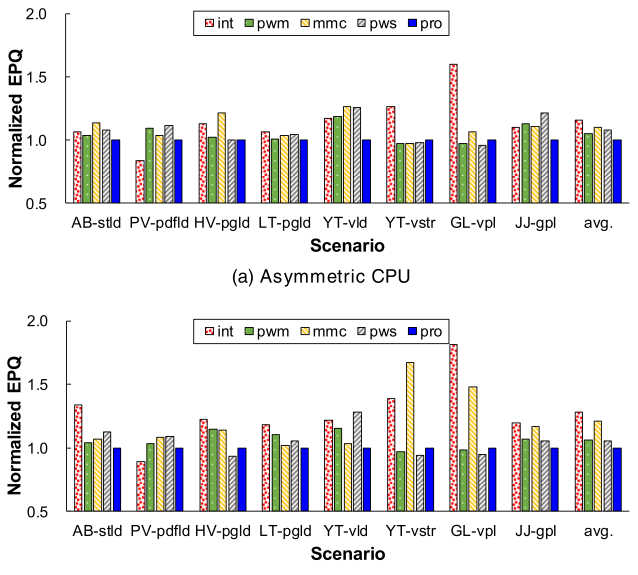


FIGURE 12. **Normalized EPQ with basic test scenarios; EPQ was normalized based on the proposed policy. Pro is proposed scenario-aware policy and others are defined in the text.**

utilization, too-frequent execution of the policy burdened the memory subsystem, and therefore, increased the overall active time of CPU cores spent for the proposed policy. Second, if the period increases, the responsiveness decreases, so performance tends to decrease. However, the performance when the period was 50 ms was lower than the performance when the period was 100 ms; this reversal of the trend was a result of coarser monitoring resolution. Considering all of these observations, we set the period of the proposed policy as 100 ms, which minimized the EPQ.

C. EVALUATION

To evaluate the effectiveness of the proposed scenario-aware policy, we compared energy, performance, and EPQ of the proposed policy with those of the benchmark methods. To reduce the measurement noise, we conducted measurements of all cases three times and took the average of the measured values.

1) BASIC TEST SCENARIOS

First, we conducted the evaluation using basic test scenarios (Table 3). To demonstrate that the proposed policy works effectively for mobile systems that use either asymmetric or symmetric multicore CPUs, we turned on and off the big CPU cluster in the target platform during the evaluation. In this experimental study, asymmetric CPU means that both big and LITTLE CPU clusters are turned on, and symmetric CPU means that only LITTLE CPU cluster is turned on.

First, we investigated EPQ to represent how polices trade-off energy and QoS numerically. Fig. 12a shows EPQ in each scenario and average EPQ in the target platform that uses asymmetric CPU. On average, the proposed policy provided the lowest EPQ, followed by *pwm* (5.4 % higher than that of the proposed policy). These results imply that under the asymmetric CPU condition, the proposed policy manages the trade-off between energy consumption and QoS the most effectively. Except for PV-pdfld, *int* provided higher EPQ, which was on average 15.5 % higher than that of the proposed policy.

Fig. 12b shows EPQ of each scenario and average EPQ in the target platform that uses symmetric CPU. Under the symmetric CPU condition, EPQ also differed among scenarios, but slightly more compared to the evaluation under the asymmetric CPU condition. Turning off the big CPU cluster caused a decrease in CPU capacity. LITTLE CPU cluster had to process all of the work including the heavy jobs that were originally processed by big CPU cluster. On average, the proposed policy showed the lowest EPQ, followed by *pws* (5.3 % higher than that of the proposed policy). On the other hand, to compensate for the decreased CPU capacity, *int* consumed too much energy, which is resulted as much higher EPQ (28.3 % higher than that of the proposed policy). This additionally implies that the proposed policy provides a good trade-off between energy and QoS even when the CPU capacity is relatively low.

Fig. 13 represents the normalized energy and performance (inverses of job delay and response latency, and FPS), under the asymmetric CPU condition. The proposed policy scaled CPU frequencies effectively to balance the trade-off between energy and performance regardless of scenario types (Fig. 13). On average, compared to *int*, the proposed policy reduced energy consumption by 25.5 % and decreased the performance by 15.2 %; the performance difference was much smaller than the energy-consumption difference. Compared to the other policies than *int*, the proposed policy provided higher performance in most scenarios. On average, compared to the proposed policy, *pwm* reduced performance by 8.1 % while consumed 3.5 % less energy. *Mmc* satisfied the required QoS and consumed low energy for FPS scenarios, but could not provide a good trade-off between energy and performance for other scenarios. *Pws* showed the worst performance because it fixed the operating frequencies at the lowest levels. When the required performance was relatively low (Figs. 13f-g; YT-vstr: 31 FPS, GL-vpl: 32 FPS), the proposed policy consumed a similar amount of energy to that consumed by *pwm* and *pws*, which tend to consume low energy. On the contrary, *int* consumed the largest amount of energy but could not increase the performance.

QoS was not always satisfied (Fig. 14). A red block means that there was a QoS violation more than once, and a blue block means that performance exceeded P_I more than once. The proposed policy provided 'just enough' performance without any QoS violation. *Int* provided the excessive performance ($> P_I$) when LT-pgld and YT-vld were executed; this means that *int* tends to provide excessive performance, and thus to waste the energy. *Mmc* and *pws* caused QoS violations when HV-pgld was running; in this case, these policies failed to satisfy end users.

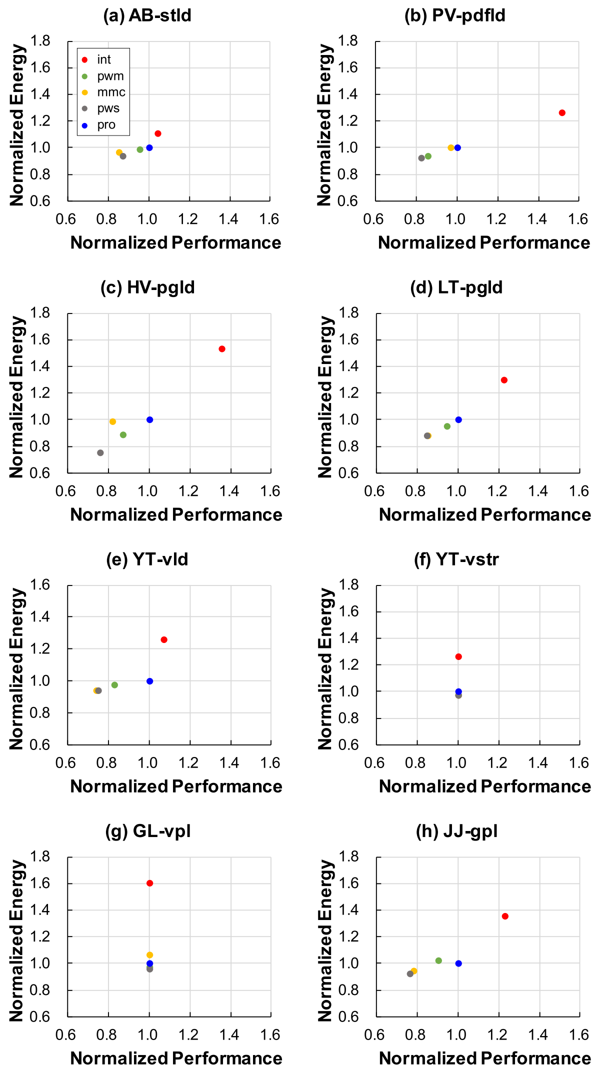


FIGURE 13. **Normalized energy and performance with basic test scenarios under the asymmetric CPU condition; energy and performance were normalized based on the proposed policy.**

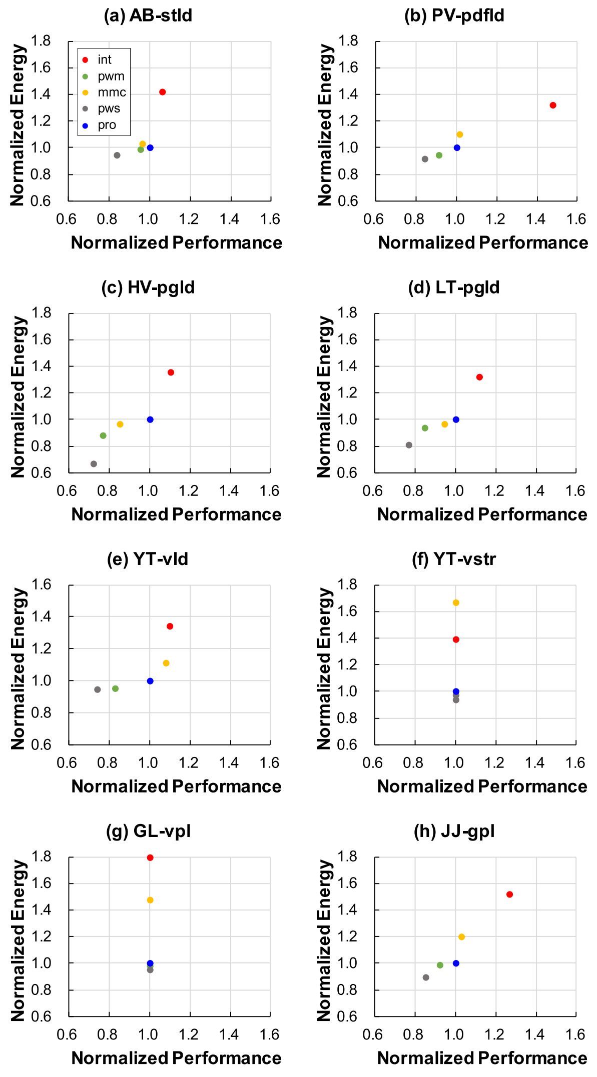


FIGURE 15. **Normalized energy and performance with basic test scenarios under the symmetric CPU condition; energy and performance were normalized based on the proposed policy.**

scenario / policy	AB-stld	PV-pdfld	HV-pgld	LT-pgld	YT-vld	YT-vstr	GL-vpl	JJ-gpl
int				🟦	🟦			
pwm								
mmc			🟥					
pws			🟥					
pro								

FIGURE 14. **QoS satisfaction with basic test scenarios under the asymmetric CPU condition (red: performance $< P_U$, blue: performance $> P_l$).**

Fig. 15 represents the normalized energy and performance under the symmetric CPU condition. The results in Fig. 15 support that for diverse scenarios, the proposed policy effectively manages energy efficiency on the mobile systems using symmetric CPU. Under the symmetric CPU condition, the proposed policy saved on average 30.7 % energy compared to *int*, while showing on average 12.3 % lower performance; the performance difference was much smaller than the energy-consumption difference. Furthermore, compared to *pwm*, the proposed policy provided 10.6 % higher performance on average while consumed only 4.7 % higher energy on average. Meanwhile, the decrease in CPU capacity degraded the quality of power management provided by *mmc* and *pwm*. Compared to the evaluation under the asymmetric CPU condition, *mmc* consumed much higher energy for FPS scenarios. Due to the decreased CPU capacity, the fluctuation degree of LITTLE CPU cluster's utilization increased, and *mmc* failed to optimize its scenario-specific coefficients. In addition, as the total CPU capacity decreased, *pwm* caused more QoS violations (Fig. 16) than did under the asymmetric CPU condition (Fig. 14).

To support the effectiveness of the proposed policy further, we investigated time-in-states of available frequency levels for basic test scenarios under the symmetric CPU condi-

scenario / policy	AB-stld	PV-pdfld	HV-pgld	LT-pgld	YT-vld	YT-vstr	GL-vpl	JJ-gpl
int								
pwm			🟥					
mmc			🟥					
pws			🟥					🟥
pro								

FIGURE 16. QoS satisfaction with basic test scenarios under the symmetric CPU condition (red: performance < P_U, blue: performance > P_l).

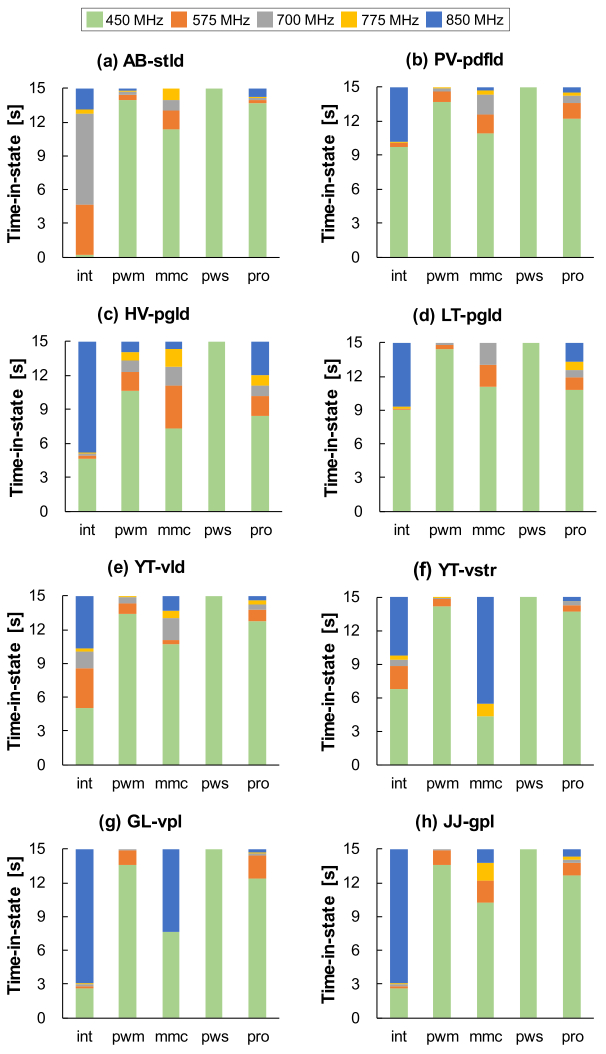


FIGURE 17. Time-in-state of available frequency levels with basic test scenarios under the symmetric condition; x-axis describes DVFS policies.

tion (Fig. 17); time-in-states indicate the proportion of time spent at each frequency level during the execution. *Int* used high-level frequencies aggressively. For instance, in Fig. 17a, *int* spent only 0.2 seconds at the lowest frequency level (450 MHz) during the execution. On the other hand, the proposed policy utilized high-level frequencies when they were necessary to process the requested work. Fig. 18 provides an example of the normalized CPU utilization and frequency

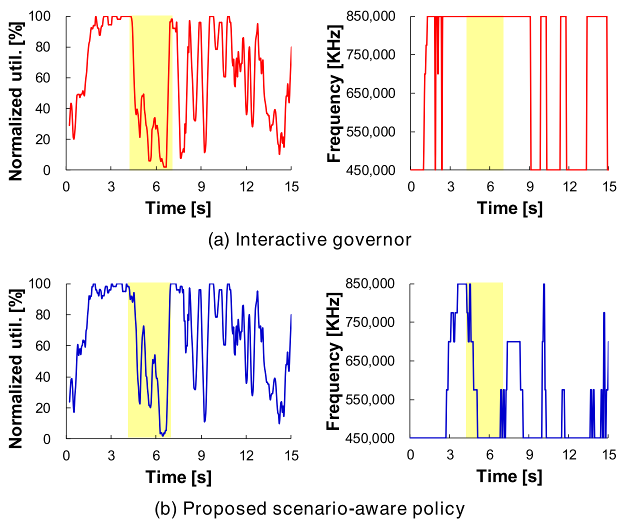


FIGURE 18. Normalized CPU utilization and frequency for HV-pgld under the symmetric CPU condition; CPU utilization was normalized based on the estimated TLP to show the average utilization for active cores.

TABLE 5. Sequential scenarios.

Scenario	Combination
Seq-1	PV-pdfld – YT-vld
Seq-2	HV-pgld – YT-vstr
Seq-3	AB-stld – HV-pgld
Seq-4	PV-pdfld – LT-pgld – GL-vpl
Seq-5	HV-pgld – YT-vld – YT-vstr
Seq-6	AB-stld – LT-pgld – YT-vld

when HV-pgld was executed; yellow region indicates the interval from page loading completion to the start of scrolling. No event occurred in this interval, so CPU utilization was relatively low, and almost zero near 6 seconds. *Int* did not scale down the frequency during this interval (Fig. 18a). This tendency of *int* degraded the energy efficiency. In contrast, the proposed method scaled down the frequency during this interval (Fig. 18b), so energy efficiency was increased significantly.

2) SEQUENTIAL SCENARIOS

In the real world, end users can switch among various user scenarios, and therefore scenario characteristics can change severely. Therefore, we conducted the additional evaluation using sequentially-combined scenarios (called sequential scenarios), under the asymmetric CPU condition. Table 5 shows sequential scenarios used for the evaluation. Each sequential scenario consists of two or three basic test scenarios. To minimize transient time from one basic scenario to another one, we initially activated all applications that correspond to the basic scenarios in the target sequential scenario. For all cases, the transient time between two basic test scenarios was less than 15 seconds. Each policy worked continuously to the end of the sequential scenario.

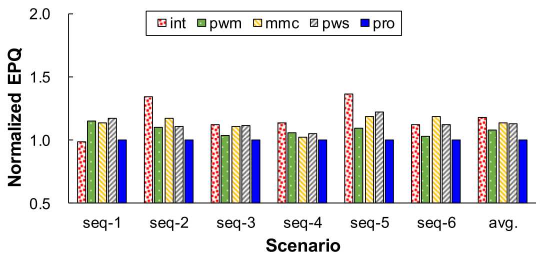


FIGURE 19. Normalized EPQ under the asymmetric architecture using sequential scenarios; EPQ was normalized based on the proposed policy.

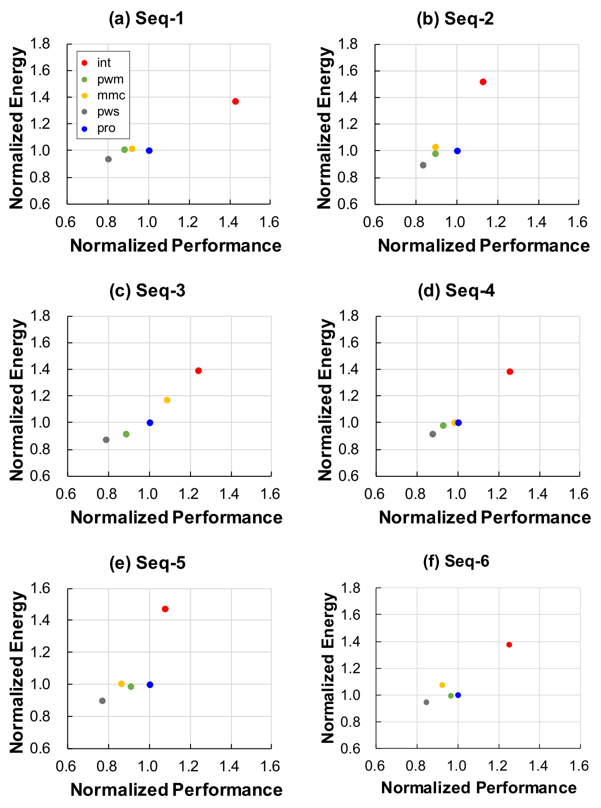
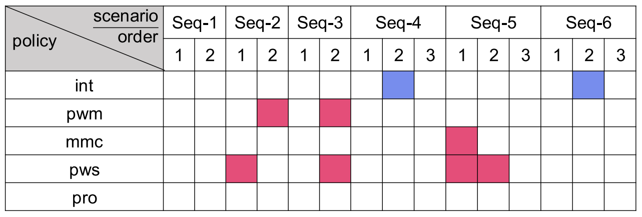


FIGURE 20. Normalized energy and performance with sequential scenarios; energy and performance were normalized based on the proposed policy.

The evaluation with sequential scenarios involved two significant differences compared to the evaluation with basic test scenarios. First, the scenario characteristic variation was larger compared to the evaluation with basic test scenarios because, generally, the variation of characteristics is larger among user scenarios than within a user scenario. Second, CPU loads increased because more than one application was active, whereas only one application was active during the execution of each basic test scenario.

The results in Figs. 19-21 imply that the proposed policy effectively adapts to the large characteristic variation and heavy CPU loads. Fig. 19 shows the normalized EPQ of each policy. When sequential scenarios were running, the proposed policy provided on average the lowest normalized EPQ. Fig. 20 shows normalized energy and performance of each



policy \ scenario, order	Seq-1		Seq-2		Seq-3		Seq-4			Seq-5			Seq-6		
	1	2	1	2	1	2	1	2	3	1	2	3	1	2	3
int								🟦						🟦	
pwm				🟥		🟥									
mmc										🟥					
pws			🟥			🟥				🟥	🟥				
pro															

FIGURE 21. QoS satisfaction with sequential scenarios (red: performance < P_U, blue: performance > P_I); 'order' means the execution order of basic test scenarios included in the corresponding sequential scenario.

TABLE 6. PCC between measured and estimated active times.

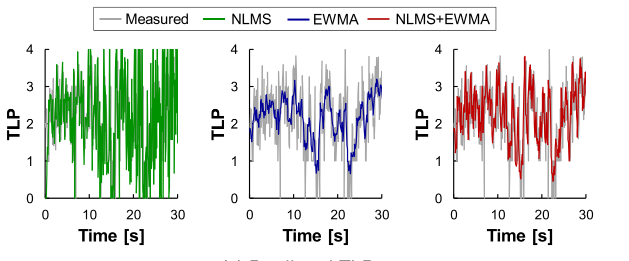
Scenarios	CPU cluster	
	big	LITTLE
AB-stld	0.998	0.867
PV-pdfld	0.998	0.968
HV-pgld	0.998	0.923
LT-pgld	0.999	0.956
YT-vld	0.999	0.912
YT-vstr	0.937	0.910
GL-vpl	0.999	0.932
JJ-gpl	0.999	0.935

policy. The proposed policy saved on average 29.6 % energy consumption compared to *int*. Besides, in most scenarios, the proposed policy provided higher performance compared to the other policies except *int*. For Seq-3, the energy and EPQ of the *mmc* were much higher than those of the proposed policy (Figs. 19 and 20c). When Seq-3 (AB-stld - HV-pgld) was executed, *mmc* used the scenario-specific coefficients for AB-stld even when HV-pgld was executed. Thus, *mmc* could not effectively adapt to the scenario changes. In Fig. 21, the proposed policy never caused any QoS violation and provided 'just enough' performance to optimize the trade-off between energy and QoS. *Int* showed excessive performance ($> P_I$) at Seq-4 and Seq-6. Besides, compared to the evaluation with basic test scenarios under the asymmetric CPU condition (Fig. 14), existing policies caused additional QoS violations due to the increase in CPU loads. *Pwm* caused QoS violations during the execution of YT-vstr (in Seq-2) and HV-pgld (in Seq-3), and *pws* caused another QoS violation during the execution of YT-vld (in Seq-5).

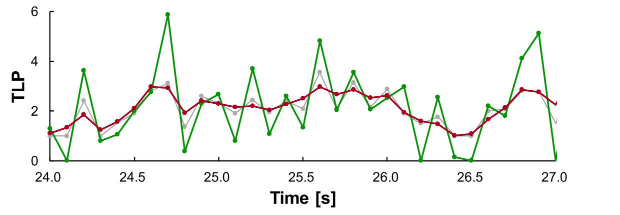
3) TLP ESTIMATION

As mentioned in Section III-A, the proposed policy estimates the fraction of active time to estimate TLP of each CPU cluster. To demonstrate the practicality of the estimation technique, we compared the measured and estimated active times of big and LITTLE CPU clusters; for this purpose, we used ATrace [37], one of the tracing utilities of Android, to trace and analyze context-switching events occurring on CPU cores.

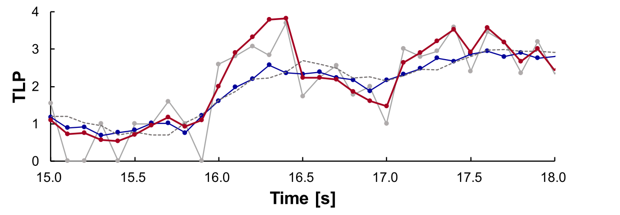
To represent the estimation accuracy numerically, we calculated the Pearson correlation coefficient (PCC) between measured and estimated active times. Table 6 shows that



(a) Predicted TLP



(b) Zoomed-in TLP (NLMS vs. NLMS + EWMA)



(c) Zoomed-in TLP (EWMA vs. NLMS + EWMA)

FIGURE 22. Part of predicted TLP (0-30 seconds) during the execution of Seq-2; a dashed line in (c) is the moving average (N = 8) of the measured TLP.

PCC values were over 0.9 except the case with LITTLE CPU cluster for AB-stld, where PCC was 0.867; these all indicate strongly-positive correlations [38]. The average PCC was 0.991 for big CPU cluster and 0.925 for LITTLE CPU cluster. Although the level of consistency can differ due to different parallelism levels among scenarios, the estimated value tended to be strongly correlated with the measured one; this means that the proposed policy successfully estimates TLP during runtime.

4) PREDICTION TECHNIQUE

To prevent energy waste and QoS degradation in advance, the proposed policy performs proactive power management by predicting scenario characteristics. As explained in Section IV-A, the proposed policy uses the combination of NLMS linear and EWMA predictors (combined predictor) to increase the prediction accuracy compared to the existing proactive approaches that use the EWMA scheme.

To verify the accuracy of the combined predictor, we predicted the upcoming TLP during the execution of Seq-2 in Table 5 (Fig. 22a). The NLMS linear predictor provided unstable prediction, and the EWMA predictor could not sharply follow peak values. Zoomed-in predicted TLPs (Fig. 22b-c) illustrate the advantages of the combined predictor compared to the NLMS linear and EWMA predictors. The combined predictor consistently gave stable prediction, whereas the NLMS linear predictor provided unstable prediction after few seconds from the start of prediction (Fig. 22b). The combined predictor could predict the measured peak values with the high accuracy, whereas the EWMA predictor could not (Fig. 22c); the predicted TLP from the EWMA predictor was similar to the moving average of the measured TLP (dashed line).

TABLE 7. Prediction error (RMSE).

Predictor	Scenarios						
	Seq-1	Seq-2	Seq-3	Seq-4	Seq-5	Seq-6	Avg.
NLMS	0.610	1.111	0.992	0.755	1.150	0.766	0.897
EWMA	0.579	0.519	0.549	0.591	0.530	0.602	0.562
NLMS+EWMA	0.416	0.327	0.354	0.457	0.345	0.439	0.390

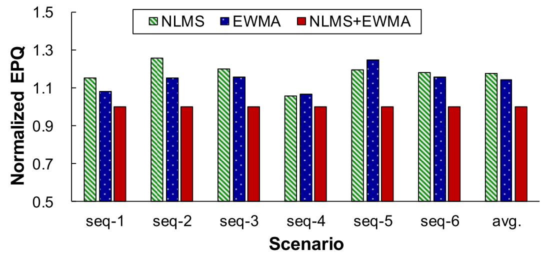


FIGURE 23. Normalized EPQ with NLMS, EWMA, and NLMS+EWMA predictors; EPQ was normalized based on the NLMS+EWMA predictor.

We measured root mean square errors (RMSEs) of the predicted TLPs obtained from the NLMS linear predictor, the EWMA predictor, and the combined predictor under the symmetric CPU condition (Table 7). The NLMS linear predictor tends to provide unstable prediction after few seconds from the start of prediction, so we used sequential scenarios that have longer execution time than the basic test scenarios. The combined predictor gave the lowest prediction error in all sequential scenarios. On average, the prediction error of the combined predictor was 56.6 % lower compared to the NLMS linear predictor, and 30.6 % lower compared to the EWMA predictor.

To demonstrate the effectiveness of the combined predictor, we measured energy consumption, performance, and EPQ for three prediction techniques. The proposed policy with the combined predictor provided the lowest EPQ (Fig. 23); this result means that the policy with the combined predictor gave the most effective balance of the trade-off between energy consumption and performance. Compared to the policy with the combined predictor, the average EPQ of the policy with the NLMS linear predictor was 17 % higher, and that of the policy with the EWMA predictor was 14 % higher. The policy with the combined predictor provided the best performance while consuming a reasonable amount of energy (Fig. 24), except for the execution of Seq-6 (Fig. 24f). On average, compared to the policy with the combined predictor, the policy with the NLMS linear predictor provided 11.8 % higher energy consumption and 2.1 % lower performance, and the policy with the EWMA predictor provided 2.1 % higher energy consumption and 8.7 % lower

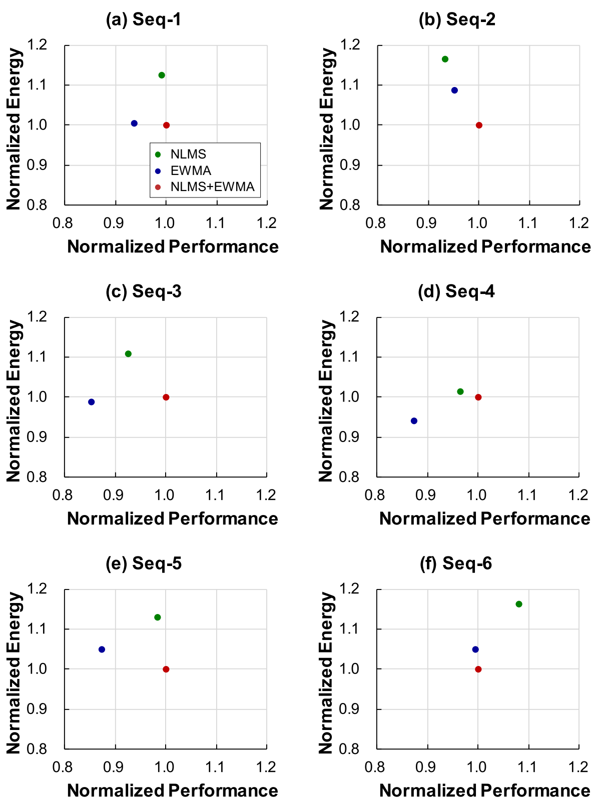


FIGURE 24. **Normalized energy and performance with NLMS, EWMA, and NLMS+EWMA predictors; energy and performance were normalized based on the NLMS+EWMA predictor.**

policy \ scenario / order	Seq-1		Seq-2		Seq-3		Seq-4			Seq-5			Seq-6		
	1	2	1	2	1	2	1	2	3	1	2	3	1	2	3
NLMS						🟥									
EWMA						🟥				🟥					
NLMS+EWMA															

FIGURE 25. **QoS satisfaction with NLMS, EWMA, and NLMS+EWMA predictors (red: performance < PU, blue: performance > P_l); 'order' means the execution order of basic test scenarios included in the corresponding sequential scenario.**

performance. Furthermore, both NLMS linear and EWMA predictors caused QoS violations, while there was no QoS violation for the policy with the combined predictor (Fig. 25). The NLMS linear and EWMA predictors failed to optimize energy efficiency and performance due to the instability of prediction and the inability of peak-value prediction.

VI. CONCLUSION

We suggested the appropriate scenario characteristics for effective power management of modern mobile devices and proposed a new scenario-aware DVFS policy that adjusts operating frequencies of CPU clusters. The proposed policy considers the parallelism level to provide appropriate processing speed for optimal energy efficiency. The proposed policy was intended for online power management of modern mobile systems. For this purpose, the proposed policy does not use any preliminary information about the target scenarios. The proposed policy manages power consumption proactively to avert energy waste, thermal emergencies, and performance degradation in advance. Using real-world user scenarios, we validated the proposed policy for mobile systems that use either asymmetric or symmetric multicore architecture-based CPUs. Compared to existing DVFS policies, the proposed policy achieved up to 25.5 % energy reduction on the mobile system that uses asymmetric CPU, and up to 30.7 % energy reduction on the mobile system that uses symmetric CPU. For all cases, the proposed policy caused no QoS violation. The experimental results demonstrated that the proposed policy successfully trades-off energy consumption and QoS regardless of user scenarios.

REFERENCES

[1] H. Chung, M. Kang, and H. Cho, "Heterogeneous multi-processing solution of Exynos 5 octa with ARM big.LITTLE technology," Samsung Electron., White Paper, 2012.

[2] M. Kim, Y. G. Kim, S. W. Chung, and C. H. Kim, "Measuring variance between smartphone energy consumption and battery life," *Computer*, vol. 47, no. 7, pp. 59–65, Jul. 2014.

[3] K.-C. Chen, E.-J. Chang, H.-T. Li, and A.-Y. Wu, "RC-based temperature prediction scheme for proactive dynamic thermal management in throttle-based 3D NoCs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 1, pp. 206–218, Jan. 2015.

[4] Y. G. Kim, J. Kong, and S. W. Chung, "A survey on recent OS-level energy management techniques for mobile processing units," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 10, pp. 2388–2401, Oct. 2018.

[5] M. Keating, *Low Power Methodology Manual for System-on-Chip Design*. New York, NY, USA: Springer, 2017, pp. 3–9.

[6] D. Brodowski, N. Golde, R. J. Wysocki, and V. Kumar. (Nov. 8, 2019). *CPUFreq Governors*. [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[7] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra, "Integrated CPU-GPU power management for 3D mobile games," in *Proc. DAC*, 2014, pp. 1–6.

[8] A. Pathania, "Power-performance modelling of mobile gaming workloads on heterogeneous MPSoCs," in *Proc. DAC*, 2015, pp. 1–6.

[9] A. Pathania, S. Pagani, M. Shafique, and J. Henkel, "Power management for mobile games on asymmetric multi-cores," in *Proc. ISLPED*, Jul. 2015, pp. 243–248.

[10] J.-G. Park, N. Dutt, and S.-S. Lim, "ML-gov: A machine learning enhanced integrated CPU-GPU DVFS governor for mobile gaming," in *Proc. ESTIMedia*, 2017, pp. 12–21.

[11] N. C. Nachiappan, P. Yedlapalli, N. Soundararajan, A. Sivasubramaniam, M. T. Kandemir, R. Iyer, and C. R. Das, "Domain knowledge based energy management in handhelds," in *Proc. HPCA*, Feb. 2015, pp. 150–160.

[12] P.-K. Chuang, Y.-S. Chen, and P.-H. Huang, "An adaptive on-line CPU-GPU governor for games on mobile devices," in *Proc. ASP-DAC*, Jan. 2017, pp. 653–658.

[13] Y. Zhu, A. Srikanth, J. Leng, and V. J. Reddi, "Exploiting Webpage characteristics for energy-efficient mobile Web browsing," *IEEE Comput. Archit. Lett.*, vol. 13, no. 1, pp. 33–36, Jan. 2014.

[14] J. Ren, L. Gao, H. Wang, and Z. Wang, "Optimise Web browsing on heterogeneous mobile platforms: A machine learning based approach," in *Proc. INFOCOM*, May 2017, pp. 1–9.

[15] J. M. Kim, M. Kim, and S. W. Chung, "Application-aware scaling governor for wearable devices," in *Proc. PATMOS*, Sep. 2014, pp. 1–8.

[16] R. A. Shafik, S. Yang, A. Das, L. A. Maeda-Nunez, G. V. Merrett, and B. M. Al-Hashimi, "Learning transfer-based adaptive energy minimization in embedded systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 6, pp. 877–890, Jun. 2016.

[17] E. Del Sozzo, G. C. Durelli, E. M. G. Trainiti, A. Miele, M. D. Santambrogio, and C. Bolchini, "Workload-aware power optimization strategy for asymmetric multiprocessors," in *Proc. DATE*, 2016, pp. 531–534.

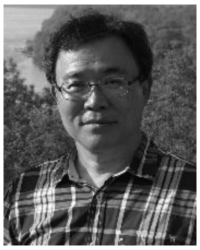
[18] K. Rao, J. Wang, S. Yalamanchili, Y. Wardi, and Y. Handong, "Application-specific performance-aware energy optimization on Android mobile devices," in *Proc. HPCA*, Feb. 2017, pp. 169–180.
[19] B. K. Reddy, G. V. Merrett, B. M. Al-Hashimi, and A. K. Singh, "Online concurrent workload classification for multi-core energy management," in *Proc. DATE*, Mar. 2018, pp. 621–624.
[20] M. J. Walker, "Run-time power estimation for mobile and embedded asymmetric multi-core CPUs," in *Proc. HiPEAC*, 2015.
[21] G. Blake, R. G. Dreslinski, T. Mudge, and K. Flautner, "Evolution of thread-level parallelism in desktop applications," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 302–313, Jun. 2010.
[22] (Jan. 15, 2020). *Kernel Timer Systems*. [Online]. Available: https://elinux.org/Kernel_Timer_Systems
[23] K. Choi, R. Soma, and M. Pedram, "Dynamic voltage and frequency scaling based on workload decomposition," in *Proc. ISLPED*, 2004, pp. 174–179.
[24] J. M. Kim, Y. G. Kim, and S. W. Chung, "Stabilizing CPU frequency and voltage for temperature-aware DVFS in mobile devices," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 286–292, Jan. 2015.
[25] F. Paterna and T. S. Rosing, "Modeling and mitigation of extra-SoC thermal coupling effects and heat transfer variations in mobile devices," in *Proc. ICCAD*, Nov. 2015, pp. 831–838.
[26] P. Eitschberger, S. Holmbacka, and J. Keller, "Trade-off between performance, fault tolerance and energy consumption in duplication-based taskgraph scheduling," in *Proc. ARCS*, 2018, pp. 3–17.
[27] B. Dietrich, D. Goswami, S. Chakraborty, A. Guha, and M. Gries, "Time series characterization of gaming workload for runtime power management," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 260–273, Jan. 2015.
[28] B. K. Reddy, A. K. Singh, D. Biswas, G. V. Merrett, and B. M. Al-Hashimi, "Inter-cluster thread-to-core mapping and DVFS on heterogeneous multi-cores," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 4, no. 3, pp. 369–382, Jul. 2018.
[29] *ARM Versatile Express Juno Development Platform, V2M-Juno Technical Reference Manual*, ARM Corp., 2014.
[30] Y. Zhu, M. Halpern, and V. J. Reddi, "Event-based scheduling for energy-efficient QoS (eQoS) in mobile Web applications," in *Proc. HPCA*, Feb. 2015, pp. 137–149.
[31] PotPlayer. *Multifunctional Media Player*. Accessed: Nov. 8, 2019. [Online]. Available: https://daumpotplayer.com
[32] *Dumpsys*. Accessed: Nov. 8, 2019. [Online]. Available: https://developer.android.com/studio/command-line/dumpsys
[33] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "RERAN: Timing- and touch-sensitive record and replay for Android," in *Proc. ICSE*, May 2013, pp. 72–81.
[34] A. Carroll and G. Heiser, "Unifying DVFS and offlining in mobile multi-cores," in *Proc. IEEE RTAS*, Apr. 2014, pp. 287–296.
[35] (Mar. 5, 2020). *Intel VTune Profiler User Guide*. Accessed: Feb. 7, 2020. [Online]. Available: https://software.intel.com/en-us/vtune-help-cpu-utilization
[36] W. Jung, "DevScope: A nonintrusive and online power analysis tool for smartphone hardware components," in *Proc. CODES+ISSS*, 2012, pp. 353–362.
[37] K. Yaghmour. (2013). *Android Platform Debugging and Development*. Accessed: Nov. 8, 2019. [Online]. Available: https://elinux.org/images/5/54/Yaghmour-android-platform-debug-dev-clean-131030.pdf
[38] K. Witz, D. E. Hinkle, W. Wiersma, and S. G. Jurs, "Applied statistics for the behavioral sciences," *J. Educ. Statist.*, vol. 15, no. 1, p. 84, 1990.

SODAM HAN (Student Member, IEEE) received the B.S. degree in computer science and electrical engineering from Handong Global University, Pohang, South Korea, in 2015, and the M.S. degree in electrical engineering from the Pohang University of Science and Technology, Pohang, in 2017, where she is currently pursuing the Ph.D. degree in electrical engineering.

Her current research interests include MPSoC design methodology, system-level power/thermal analysis and management, and low power and thermal-aware system design.

YONGHEE YUN received the B.S. degree in electronics and electrical engineering from Dankook University, Yongin, South Korea, in 2014, and the Ph.D. degree in electrical engineering from the Pohang University of Science and Technology, Pohang, South Korea, in 2019.

His current research interests include MPSoC design methodology, and system-level power/thermal analysis and management.



YOUNG HWAN KIM (Senior Member, IEEE) received the B.E. degree in electronics from Kyungpook National University, South Korea, in 1977, and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Berkeley, Berkeley, CA, USA, in 1985 and 1988, respectively.

From 1977 to 1982, he was with the Agency for Defense Development, South Korea, where he was involved in various military research projects, including the development of autopilot guidance and control systems. From 1983 to 1988, he was a Postgraduate Researcher with the Electronic Research Laboratory, University of California at Berkeley, where he was involved in developing VLSI CAD programs. He is currently a Professor with the Division of Electronic and Computer Engineering, POSTECH, South Korea. His research interests include plasma and liquid crystal display systems, multimedia circuit design, MPSoC and GPGPU system design for display and computer vision applications, statistical analysis and design technology for deep-submicron semiconductor devices, and power noise analysis.

Dr. Kim has served as an Editor for the *Journal of the Institute of Electronics Engineers of Korea*, and as the General Chair and a Committee Member of various Korean domestic and international technical conferences, including the International SoC Design Conference, the IEEE ISCAS 2012, and the IEEE APCCAS 2016.

SEOKHYEONG KANG (Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the Pohang University of Science and Technology, Pohang, South Korea, in 1999 and 2001, respectively, and the Ph.D. degree in computer engineering from the University of California at San Diego (UCSD), La Jolla, CA, USA, in 2013.

He was with the System-on-Chip (SoC) Development Team, Samsung Electronics, Suwon, South Korea, from 2001 to 2008, where he was involved in development and commercialization of multimedia SoC. He was with the Department of Electrical Engineering, Ulsan National Institute of Science and Technology, Ulsan, South Korea, from 2014 to 2018. Since 2018, he has been with the Department of Electrical Engineering, Pohang University of Science and Technology. His current research interests include low power design optimization and cost-driven methodology for chip implementation.