IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 2, 2019, accepted December 26, 2019, date of publication January 10, 2020, date of current version January 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2965548

Performance Improvement of Linux CPU Scheduler Using Policy Gradient Reinforcement Learning for Android Smartphones

JUNYEONG HAN AND SUNGYOUNG LEE
LG Electronics, Seoul 07336, South Korea
Corresponding author: Junyeong Han (junyeong.han@lge.com)

ABSTRACT The Energy Aware Scheduler (EAS) was developed and applied to the Linux kernel of recent Android smartphones in order to exploit the ARM big.LITTLE processing architecture efficiently. EAS organizes CPU hardware information into Energy Model which are used to improve CPU scheduling performance. In particular, it reduces power consumption and improves process scheduling performance. However, EAS has limitations in improving CPU scheduling performance, because the Energy Model configures the CPU hardware information to fixed values, which does not reflect the characteristics of running tasks, such as the workload changes and the transition between running state and sleep state. To solve this problem, this paper introduces the Learning Energy Aware Scheduler (Learning EAS). The Learning EAS adjusts the TARGET_LOAD used to set the CPU frequency and the sched_migration_cost used as the task migration criteria according to the characteristics of the running task through the policy gradient reinforcement learning. In LG G8 ThinQ, Learning EAS improved power consumption by 2.3% - 5.7%, hackbench results for process scheduling performance by 2.8% - 25.5%, applications entry time by 4.4% - 6.1%, and applications entry time under high CPU workload by 9.6% - 12.5%, respectively compared with EAS. This paper also showed that the Learning EAS is scalable by applying the Learning EAS to high-end and low-end chipset platforms of Qualcomm.Inc and MediaTek.Inc and improving power consumption by 2.8% - 7.8%, application entry time by 2.2% - 7.2%, respectively compared with EAS. Finally, this paper showed that the performance of CPU scheduling is improved gradually by the repetition of reinforcement learning.

INDEX TERMS ARM big.LITTLE processing architecture, energy aware scheduler, process scheduler, CPU frequency governor, reinforcement learning, policy gradient, neural network, power consumption.

I. INTRODUCTION

Customers choose a smartphone by comparing the smartphone's performance and battery life as major criteria [1]. Since the smartphone's CPU is one of the significant hardware that affects both performance and battery life, smartphone chipset vendors such as Qualcomm.Inc and MediaTek.Inc release chipset platforms each year with new CPUs which have higher performance and lower power consumption to satisfy customer needs. For Android smartphones, chipset vendors distribute Android that change the Linux process scheduler or add CPU control function to make efficient use of the new CPU with chipset platforms.

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

ARM.Ltd introduced the big.LITTLE technology [2] focusing on patterns of changing CPU workload depending on the running applications. The big.LITTLE technology uses only low-power LITTLE CPU when the CPU workload is low, and uses high-performance big CPU additionally when the CPU workload is high, thus increasing the battery life in the mobile environment. Chipset platforms with a combination of various big and LITTLE processors were developed by smartphone chipset vendors and CPU schedulers were also developed to use the big.LITTLE processing architecture efficiently.

Linux CPU schedulers, such as Global Task Scheduler (GTS) [2], Energy Aware Scheduler (EAS) [3], and Hot Plug Strategy (HPS) [4], allocate tasks to the appropriate CPU based on the CPU workload and determine the

CPU frequency. However, because the CPU workload, which is basis on CPU scheduling, changes greatly every tens of milliseconds, it is not accurate to predict the future CPU workload with the current CPU workload. Therefore, the current Linux CPU scheduler's approach has limitations in handling the CPU workload of applications with CPU resources of minimal power consumption.

In the Linux CPU scheduler for the big.LITTLE processing architecture, process scheduling performance is most affected by task migration efficiency, because the task migration assigns tasks to the proper CPU for the use of CPU resources efficiently. Currently, most Linux CPU schedulers perform task migration through load balancing [5] in the Linux process scheduler. However, this load balancing does not consider the characteristics of the transition between running state and sleep state of each running task, and only performs task migration when the utility of task migration is larger than the fixed task migration cost, so the process scheduling performance may be degraded depending on the running tasks.

To improve these two problems, this paper proposes a Learning EAS that reduces power consumption and improves process scheduling performance by applying policy gradient reinforcement learning [6] to the EAS, the latest Linux CPU scheduler. Learning EAS reduces power consumption by providing a minimum CPU frequency that can process changing CPU workloads according to the workload change of running tasks. Learning EAS also improves process scheduling performance by maximizing task migration efficiency, adjusting task migration cost according to the characteristics of the transition between running state and sleep state of running tasks.

The paper is organized as follows. Section 2 compares the existing Linux CPU scheduling techniques and explains the limitations. Section 3 describes the Learning EAS applying the policy gradient reinforcement learning technique to achieve optimal CPU scheduling performance goals. Section 4 compares the power consumption and performance of EAS and Learning EAS to explain the effect of Learning EAS. Finally, Section 5 describes future research direction.

II. RELATED WORK

The first ARM big.LITTLE processing pair consists of the Cortex-A15 [7] and Coretex-A7 [8] processors implementing the AMRv7-A [9] architecture. The key idea of big.LITTLE technology is to use the Cortex-A15 processor only when high performance is required and to use the Cortex-A7 processor elsewhere to process CPU workload with minimal power consumption. Fig.1 [10] is a graph showing the correlation between performance and power consumption of big and LITTLE core in a typical big.LITTLE chipset platform. This graph represents the form of quadratic equations.

The basic view of ARM big.LITTLE technology is to allow software to see big and LITTLE processors in the same architecture. For example, on the big.LITTLE processor, which is a combination of four big CPUs and four LITTLE CPUs,

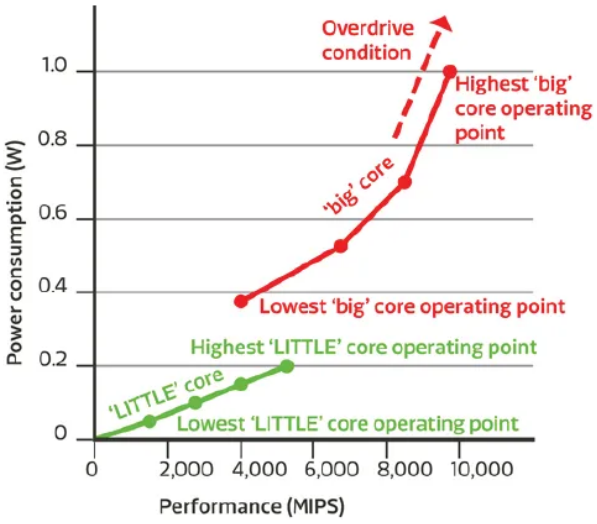

FIGURE 1. Graph of correlation between power consumption and performance.

the Linux kernel can control each CPU with the same software interface from cpu0 to cpu7. Taking advantage of this, ARM proposed a Global Task Scheduler (GTS) with Linaro that can efficiently use the big.LITTLE processor.

GTS knows the CPU workload of an individual task and allocates the task to the CPU with the lowest power among the CPUs that can address that task's workload. GTS divides runtime behavior into three categories: high-intensity workloads, sustained workloads, and long-use and low-intensity workloads. In the case of high-intensity workloads, the workloads are bursty, so GTS allocates tasks to big CPUs when workloads are high and LITTLE CPUs when workloads are low. If workloads change very fast, the task migration cost between big and LITTLE CPUs increases, so GTS only uses big CPUs. In the case of sustained workloads, GTS assigns the task to the LITTLE CPU running at the maximum frequency, and assigns the task to the big CPU as needed. For long-use, low-intensity workloads, only the LITTLE CPU is used, so the benefits for power consumption are clear. Since dividing categories according to workloads is determined by the tuning value, tuning is required for each big.LITTLE processor.

The Energy Aware Scheduler (EAS), developed by ARM, Linaro, and Qualcomm, works basically like GTS, but schedules tasks through the Energy Model [11]. The Energy Model provides hardware information, such as power consumption and performance of the cluster and the CPUs included in that cluster to distribute tasks to the proper CPU and determine that CPU frequency to minimize power consumption. EAS eliminates tuning values and minimizes power consumption for each chipset platform by providing accurate information on each chipset platform through the Energy Model. EAS has been applied to Google Pixel2 [12] with Snapdragon 835 chipset platform [13] which consist four Kryo 280 CPUs based on ARM Cortex-A73 [14] as big processors and four

Kryo 280 CPUs based on ARM Cortex-A53 [15] as LITTLE processors.

MediaTek developed their own unique Hot Plug Strategy (HPS) for MT6750T/MT6750S chipset platform [16], which consist four Cortex-A53 CPUs supporting 1.5 GHz frequency as big processors and four same CPUs supporting 1.0 GHz frequency as LITTLE processors. HPS calculates each CPU workload every 40ms and keeps only the minimum number of CPUs online to process its CPU workload requirements to minimize power consumption. Recently, most MediaTek chipset platforms use customized EAS.

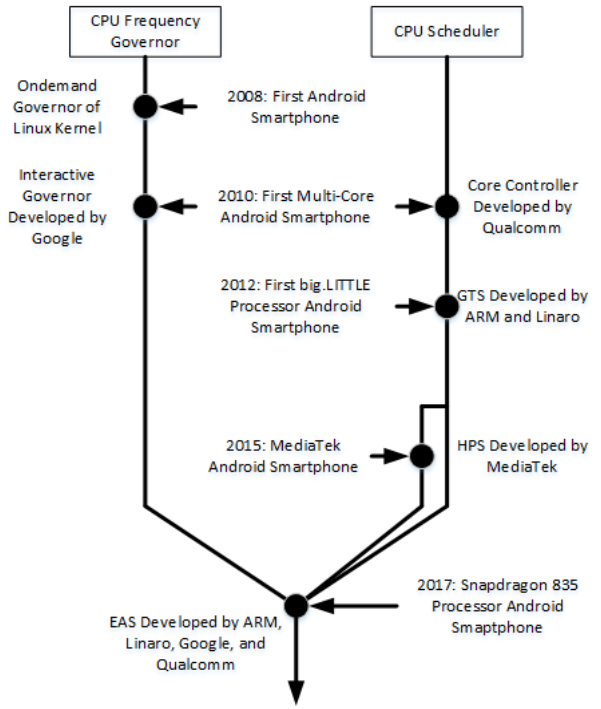

FIGURE 2. Advances in CPU Governor and CPU Scheduler as Processor Development in Android Smartphone.

Fig. 2 shows the evolution of the CPU frequency governor and the CPU scheduler with the development of processors in Android smartphones. As the CPU frequency governor to determine the CPU frequency, the ondemand governor [17] was mainly used in a single core architecture, and the interactive governor [18] was primarily used in a multi core architecture. GTS and HPS were used as CPU schedulers to determine the number of online CPUs or the number of active CPUs. Finally, in 2017, EAS which integrates the CPU frequency governor was developed.

Kim *et al.* classify an application into multimedia application category and other categories and allocates the multimedia application to LITTLE CPUs [19]. This scheme reduces the power consumption when using multimedia applications. Energy-aware CPU Frequency Scaling (EFS) [20] reduces the power consumption on YouTube by setting the CPU frequency optimized for data transfer in video streaming applications. However, such research reduces the power consumption only in a specific category of applications.

Datta and Patel improves CPU scheduling performance by using the Cache Miss Priority CPU Scheduler (CM-PCS) algorithm and the Context Switch Priority CPU Scheduler (CS-PCS) algorithm [21]. And Feliu *et al.* improves scheduling performance through process scheduling considering the bandwidth of L1 cache and main memory and scheduling fairness based on the experience data of the process [22]. However, the utility of such research has diminished because CPU scheduling works based on precise information about task migration cost in a CPU cluster and inter clusters considering L2 cache through Energy Model in EAS.

EAS generally provides good performance because many of the values which are basis for the operation are fixed, but EAS does not show the best performance regarding the workload changes and the transition between running state and sleep state of the running tasks. In general, each task has its own characteristics for changing workload. It also has unique properties for transition between running state and sleep state. For this reason, it is difficult to optimize the performance by using simple algorithm which changes the default value for the EAS operation when a task's workload or the ratio of sleep and running states gets to change above or below a certain value.

Therefore it is necessary to set the values of the EAS operation criteria dynamically according to the characteristics of the running tasks to improve performance, which can be achieved through policy gradient reinforcement learning. Policy gradient reinforcement learning is a method of learning a policy function that returns an action that maximizes the reward in a given state. As shown in Equation (1), the policy function is trained by updating parameter vector θ_{t+1}, which is changed by the product of the learning rate α and the gradient of the policy performance measure ρ by parameter vector θ_t.

$$\theta_{t+1} = \theta_t + \alpha \frac{\partial \rho}{\partial \theta_t} \tag{1}$$

The policy gradient reinforcement learning implemented by artificial neural network for the simulated electric power trade shows the result of maximizing the performance [23]. In addition, a study that reduced CPU power consumption in mobile environment by learning through artificial neural network was introduced [24] and showed the possibility of OS optimization through machine learning.

III. LEARNING ENERGY AWARE SCHEDULER

In this paper, we designed and implemented Learning Energy Aware Scheduler (Learning EAS) to achieve performance closer to the theoretical optimal CPU scheduler than EAS. This section consists of seven subsections, from A to G. And this section details the theoretical background, design, and implementation for the Learning EAS. Subsection A defines the theoretical optimal CPU scheduler behavior by expressions. These expressions are used to implement the Learning EAS mentioned in subsection D and F. Subsection B describes the overall structure and behavior of the original

EAS on a big.LITTLE chipset platform. Subsection C describes schedutil CPU frequency governor which is operated on a big.LITTLE chipset platform and subsection D describes the dynamic change of the TARGET_LOAD in the schedutil using policy gradient reinforcement learning. Subsection E describes task migration in process scheduler which is operated on a big.LITTLE chipset platform, and subsection F describes the dynamic change sched_migration_cost for task migration using policy gradient reinforcement learning. Finally, subsection G explains the implementation of Learning EAS.

A. THEORETICAL OPTIMAL CPU SCHEDULER

Tasks have a workload, which is a set of instructions to be processed through the CPU. In the Linux kernel, when a task does not use any CPU resources in a unit of time, the workload of this task is 0, and when a task uses one CPU resource entirely, the workload of this task is defined as 100. Allocating less CPU resources than a task workload delays processing of the workload, resulting in a long response time. For example, if a real-time game program fails to generate a frame within a given time for a target FPS(Frame Per Second), the user may feel dropped or delayed frames. Therefore, the optimal CPU scheduler is to configure the CPU resources to minimize the power consumption whereas providing CPU resources to process the workload of all running tasks.

In the Linux kernel, the total workload of tasks running on a single CPU becomes that CPU workload, so task workload and CPU workload can be represented by the expressions of relation. Task workload, CPU workload, and CPU throughput are all in the same unit and can be compared because one CPU throughput is the maximum CPU workload that the CPU can process. Since CPU throughput is directly proportional to the frequency at which the CPU operates, the maximum CPU throughput can be defined as max CPU frequency (Hz). The minimum CPU throughput can be defined as CPU idle, which is 0 frequency (Hz) because it does not process CPU workload. Therefore, one CPU throughput has a value between 0 (Hz) and max CPU frequency (Hz).

Given the workload of all running tasks, it is possible to calculate the CPU workload required to process the workload of these tasks and the CPU resource with the minimum power consumption to process this CPU workload. Information about a system that *Ntask* tasks are executed in a processor consist of *Ncpu* CPUs is defined as shown in Table 1.

CPU workload and CPU throughput are defined by Equation (2) and (3), respectively.

$$CPU\ workload = \sum_{i=1}^{Ntask} W_i \tag{2}$$

$$CPU\ throughput = \sum_{j=1}^{Ncpu} T_j \tag{3}$$

TABLE 1. Definition of parameters related to big.LITTLE processing.

Symbol	Definition
$Ntask$	the number of running tasks
i	running task index ($1 \le i \le Ntask$)
W_i	workload of $Task_i$
$Ncpu$	the number of CPUs
j	CPU index ($1 \le j \le Ncpu$)
T_j	throughput of cpu_j
$P(T_j)$	power consumption of T_j

In the case of Inequality (4) in relation to CPU workload and CPU throughput, the optimal CPU scheduler must convert the state to satisfy the Inequality (5) by increasing the number of active CPUs and increasing the CPU frequency.

$$\sum_{i=1}^{Ntask} W_i > \sum_{j=1}^{Ncpu} T_j \tag{4}$$

$$\sum_{i=1}^{Ntask} W_i \le \sum_{j=1}^{Ncpu} T_j \tag{5}$$

And each task's workload must be processed without delay by satisfying Inequality (5) and minimizing power consumption at the same time, it can be formulated by (6) where T_j is the control variable.

$$\min_{T_j} \sum_{j=1}^{Ncpu} P\left(T_j\right) \quad subject\ to \sum_{i=1}^{Ntask} W_i \le \sum_{j=1}^{Ncpu} T_j \tag{6}$$

As shown in Fig. 1, in most CPUs the relationship between CPU frequency and power consumption is represented by the quadratic as Polynomial (7), so keeping the CPU frequency low and increasing the number of active CPUs is advantageous for power consumption.

$$P\left(T_j\right) \approx a \cdot \left(T_j\right)^2 + b \quad where\ a\ and\ b\ are\ constants \tag{7}$$

For example, if a system has two tasks each with 500 MHz workload and two CPUs each with throughput greater than 1000 MHz, the CPU workload is 1000MHz by Equation (2), and the CPU throughput must be greater than or equal to 1000 MHz according to Inequality (5). One method of setting CPU throughput to 1000 MHz is to set one CPU frequency to 1000 MHz, and the other is to set two CPU frequencies to 500 MHz. Assuming that the power consumption of one 500 MHz CPU is 400 mW and one 1000 MHz CPU is 1000 mW and the power consumption increases in direct proportion to the number of active CPUs, the power consumption of the first method is 800 mW and second is 1000 mW. Therefore, the optimal CPU scheduler sets the two CPU frequencies to 500 MHz by Formulation (6).

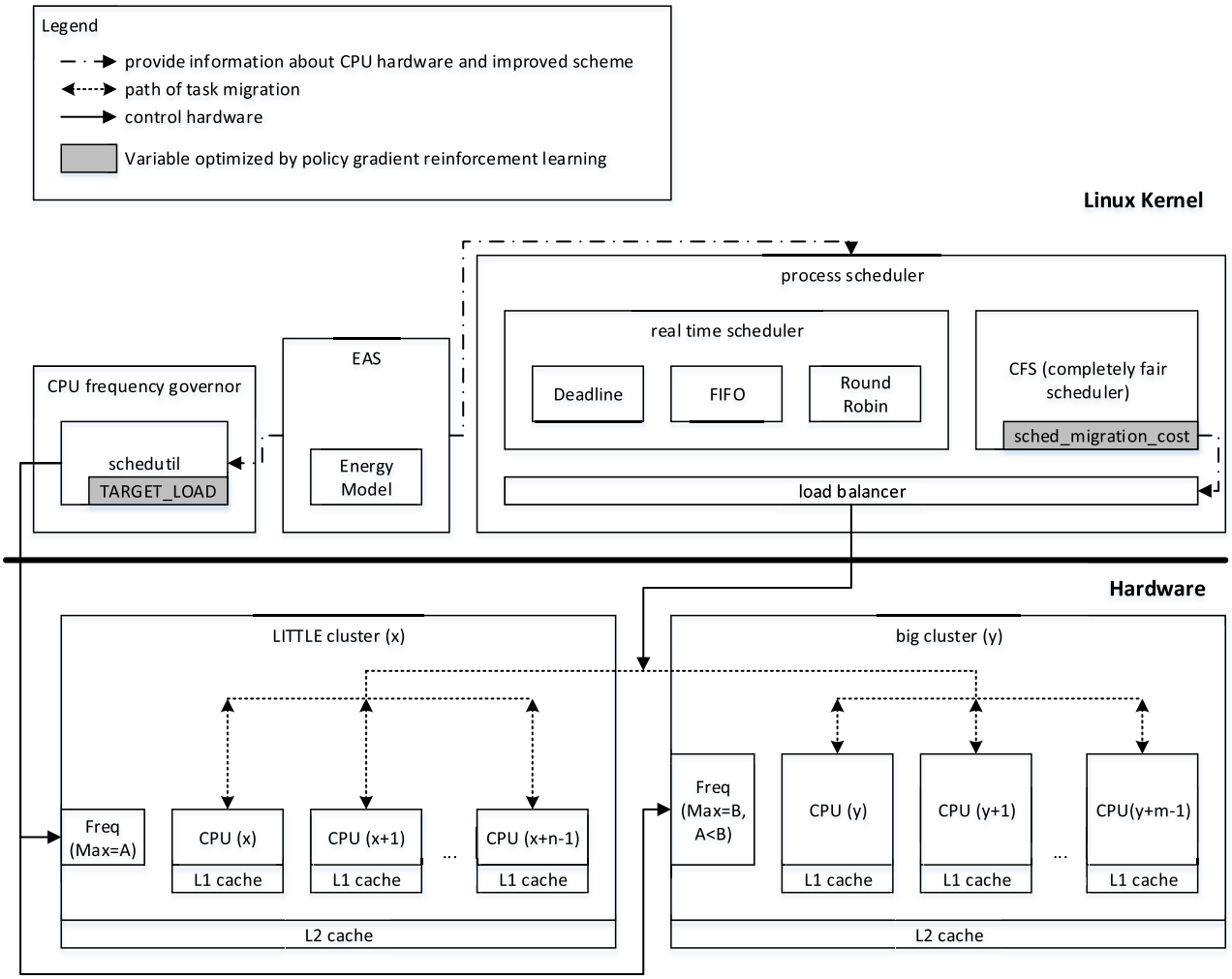

FIGURE 3. Block diagram of EAS and big.LITTLE chipset platform.

B. EAS BEHAVIOR ON THE BIG.LITTLE CHIPSET PLATFORM

Most chipset platforms currently used in Android smartphones have multiple CPUs. Usually several CPUs are grouped into a cluster, where the CPUs in the cluster share the frequency and L2 cache, with two or more clusters exist on one chipset platform. The CPU has an online state and an offline state, and the CPU of recent chipset platforms always stays online unless chipset platforms are in sleep mode. If there is a running task on the CPU, the CPU processes the task workload and if there are no remaining tasks to be run anymore, it executes the idle task [25] and goes to the idle state to minimize the power consumption.

Fig. 3 shows an arbitrary chipset platform in which n CPUs in order from x*th* CPU make up cluster (x), and m CPUs in order from y*th* CPU make up cluster (y), and EAS operating on this chipset platform.

When the max frequency of cluster (y) is higher than cluster (x), cluster (x) becomes a LITTLE cluster and cluster (y) becomes a big cluster. Traditionally, the CPU scheduler consists of the process scheduler, the core controller, and the CPU frequency governor. As the chipset platform keeps the CPU online, the core controller function is replaced by the process scheduler's load balancer, so the CPU scheduler consists of two functions: the process scheduler and the CPU frequency governor.

EAS's Energy Model improves efficiency of task migration by providing information such as CPU specific performance and task migration costs between CPUs to existing process schedulers. The Energy Model also reduces power consumption by providing information such as power consumption and heat generation according to each CPU frequency to the CPU frequency governor.

C. SCHEDUTIL CPU FREQUENCY GOVERNRO OF EAS

The schedutil [26] CPU frequency governor included in the CPU scheduler measures each task's workload and the workload of CPU on which the task was executed more accurately than the conventional CPU frequency governor, such as ondemand and interactive which operates independently

of the process scheduler. Since schedutil sets the next CPU frequency based on this exact CPU workload, it is possible to set the minimum power frequency without performance degradation when the future CPU workload does not change.

In schedutil, each CPU workload is expressed as CPU util. CPU time is the amount of time for which a CPU was used for processing that CPU workload and CPU idle time is the remaining time except CPU time. The max CPU util is the CPU util at which the CPU processes the CPU workload without CPU idle time at the maximum CPU frequency. When CPU util is *Cutil*, CPU time is *Ctime*, CPU idle time is *Itime*, and max CPU util is *MCutil*, CPU util is defined as the percentage of CPU time for a unit of time based on max CPU util, as shown in Equation (8).

$$Cutil = (\frac{Ctime}{Ctime + Itime}) \cdot MCutil \tag{8}$$

For example, when the max CPU util value is 200, the CPU time is 10ms and the CPU idle time is at 10ms, then the CPU util is 100. In this situation, if the CPU frequency is lowered instead of maintaining the CPU idle time, the CPU idle time will decrease and CPU time will increase. This results in lower power consumption of CPU. So schedutil processes CPU workload equivalent to CPU util 100 without CPU idle time at the half of max CPU frequency. In General, max CPU util of each CPU is defined by the Energy Model and the max CPU util of big CPU is set higher than the LITTLE CPU.

Since the CPUs in a cluster share the same CPU frequency, schedutil selects the largest CPU util among the CPUs in the cluster and sets the next CPU frequency as the ratio of the selected CPU util to the max CPU util based on the max CPU frequency. When the next CPU frequency is *NCfreq*, the largest CPU util in the cluster is *LCutil* and max CPU frequency is *MCfreq*, next CPU frequency is calculated as shown in Equation (9). The reason for using the largest CPU util in the cluster is to guarantee the performance of a large workload because one task's workload cannot be divided to be process by two or more CPUs. Since max CPU util, max CPU frequency, and 1.25 are constants, the next CPU frequency is determined by the maximum CPU util in the cluster.

$$NCfreq = (\frac{LCutil}{MCutil}) \cdot MCfreq \cdot (1.25) \tag{9}$$

The reason why the ratio is set to 125% is to secure a 20% performance margin using the CPU at 80% of the set CPU frequency as shown in Equation (10), so TARGET_LOAD of Fig. 2 is 80%.

$$NCfreq \cdot (0.8) = (\frac{LCutil}{MCutil}) \cdot MCfreq \tag{10}$$

D. SCHEDUTIL CPU FREQUENCY GOVERNOR ENHANCEMENT BY POLICY GRADIENT REINFORCEMENT LEARNING

Maintaining TARGET_LOAD at 0.8 satisfies Inequality (5) even if the future CPU workload is 25% higher than the current CPU workload, but minimizing the power consumption of Formulation (6) cannot be achieved. If the future TARGET_LOAD can be accurately predicted by considering the workload change patterns of running tasks, the power consumption of Formulation (6) can be further minimized. Therefore, in this paper, we propose the first function of the Learning EAS that dynamically sets the optimal TARGET_LOAD value by applying policy gradient reinforcement learning [6] which directly returns the policy with a small amount of computation considering the resource constrained mobile environment among the reinforcement learning schemes.

In order to apply reinforcement learning, function approximators such as neural networks, decision-trees, or instance-based methods should be used. In policy gradient reinforcement learning, stochastic policy is approximated directly using an independent function with the parameters as shown in Equation (1), instead of approximating the value function. According to "Theorem 2 (Policy gradient with Function approximation)" [6], the policy gradient of Equation (1) is expressed as Equation (11).

$$\frac{\partial \rho}{\partial \theta} = \sum_s d^{\pi}(s) \sum_a \frac{\partial \pi\ (s, a)}{\partial \theta} f_w(s, a) \tag{11}$$

d^{π} is the stationary distribution of states under π, which is independent of s_0 for all policies, and f_w is approximation of quality function. State s is each CPU util and a is the action to increase or decrease TARGET_LOAD. There are two typical cases where only one CPU util is high or most CPU utils are similar in the cluster according to running tasks. In the former case, even if a new task is executed, the task can be allocated to the CPU which has another margin, and in the latter case, the processing of the new task may be delayed. Therefore, the larger the size of each CPU util, the lower the TARGET_LOAD should be in order to prepare for the execution of a new task or in case a running task requires a lot of workload.

As a result of a that increases TARGET_LOAD, delays in task execution may occur because future util increases, or tasks may be processed normally because the future util is the same or decreases. In the former case, the policy function must receive a negative reward and in the latter case, must receive a positive reward. Conversely, as a result of a decreases TARGET_LOAD, delays in task execution can be reduced, even if util increases, or power consumption can be wasted because util is the same or decreases. In the former case, the policy function must receive a positive reward and in the latter case, must receive a negative reward. Since the CPU frequency is determined according to the CPU util, the reward size is set by the amount of change in the CPU frequency.

Learning EAS sets d^{π} and f_w to fixed values obtained by the power consumption test. The parameter θ of the policy function is the policy that increases or decreases TARGET_LOAD which affects the next CPU frequency change. Thus when $\pi\ (s, a)$ is partially differentiated against θ, it is the rate of the CPU frequency change.

When the current TARGET_LOAD change value is tl_t, the current CPU frequency is $freq_t$, and the previous CPU frequency is $freq_{t-1}$, for the time t in Equation (1), Equation (11) for reward can be converted into Equation (12).

$$\frac{\partial \rho}{\partial \theta_t} = tl_t(freq_t - freq_{t-1}) \tag{12}$$

When there is a cluster (x) consisting of n CPUs from the x*th* CPU, the Learning EAS implements the policy function as a policy neural network that uses all CPU utils as inputs and the policy which increases or decreases TARGET_LOAD as an output, as shown in Fig. 4.

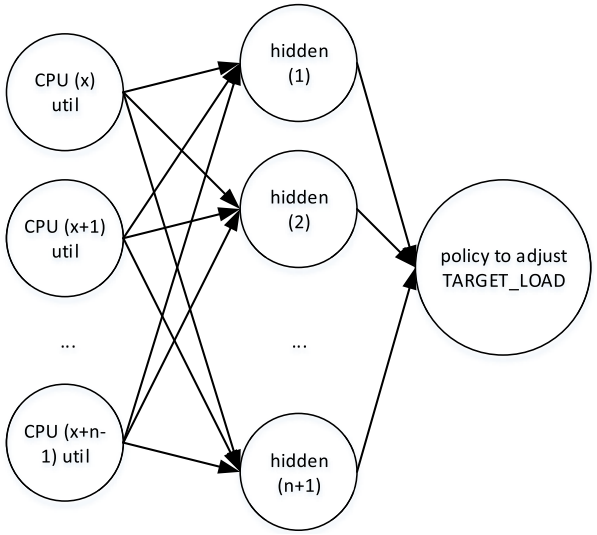

FIGURE 4. Configuration of policy neural network to set TARGET_LOAD for cluster (x).

The policy neural network has one hidden layer to minimize computations, and the hidden layer consists of (n + 1) neurons that is one more than the number of input neurons and uses ReLU for activation function.

As shown in Fig. 5, the calculation result of Equation (12) is entered as an error so that the policy function is learned through backpropagation, and the learning result is updated as synapses change of the policy neural network.

Algorithm 1 is pseudocode to set TARGET_LOAD for cluster (x) by running and training the policy neural network.

The time t is set to 100 milliseconds so that the policy neural network is received CPU utils every 100 milliseconds and executed. The result of running policy neural network is an output value between 0 and 100000. If the output is greater than the threshold 12000 which is determined by tuning, the policy decreases TARGET_LOAD and if it is less, increases TARGET_LOAD.

If the CPU util is large, it means that there are a lot of CPU workloads. Usually, if there are a lot of CPU workloads at this time, it is likely that there will be a lot of CPU workloads at the next time. So if the CPU util is large, schedutil should lower TARGET_LOAD to prepare for processing the next a lot of CPU workloads. If the CPU util used as input value in the policy neural network is large, the output value will be large, so the Learning EAS will lower TARGET_LOAD if the output value is greater than the threshold.

TARGET_LOAD should be changed as little as possible every 100 milliseconds to ensure the stability of the system. However, if it is changed too little for stability, it is difficult to minimize power consumption in response to changes in

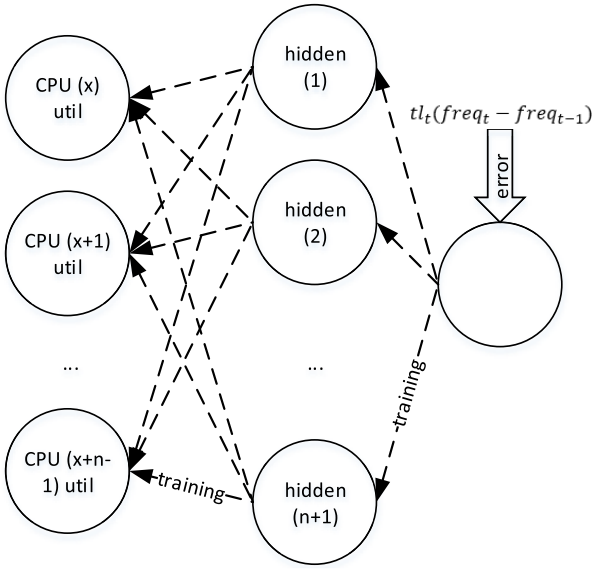

FIGURE 5. Training of policy neural network to set TARGET_LOAD for cluster (x).

Algorithm 1 Pseudocode to set TARGET_LOAD for Cluster (x) Using Policy Gradient Reinforcement Learning

```
Initialize policy neural network
CPU_utils[n] = {0, 0, ...,0}
prev_CPU_freq = cur_CPU_freq = 0
repeat
  i = 0
  while i < n do
    CPU_utils[i] = Get ith CPU load
    ++i
  end
  result = Run policy neural network(CPU_utils)
  if (threshold(12000) < result) then
    tl = -0.0023
  else then
    tl = 0.0023
  end
  TARGET_LOAD += tl
  prev_CPU_freq = cur_CPU_freq
  Sleep 100ms
  Get cur_CPU_freq
  error = tl * (cur_CPU_freq – prev_CPU_freq)
  Training policy neural network by back
  propagation(error)
until system has been shutdown
```

the running applications. When testing with changing running applications, the best power-saving effect was achieved when TARGET_LOAD was changed to 0.0023 size. So the Learning EAS changes TARGET_LOAD by 0.0023 every 100 milliseconds.

The policy neural network is trained by the calculated error with CPU frequency after the next 100 milliseconds. The policy neural network is trained by the CPU utils, which is acquired every 100 milliseconds, from system boot to shutdown. The learning results are periodically stored in the secondary repository and restored to initial values of the synapses of the policy neural network at the next system boot. The learning rate corresponding to α in Equation (1) is adjusted so that the policy neural network stabilizes within 5 minutes when the workload pattern of the running tasks changes, which is in order to keep pace with the speed of change in running applications in the mobile environment.

MediaTek uses schedplus as a CPU frequency governor instead of schedutil. Because schedplus behavior is very similar to schedutil, policy gradient reinforcement learning can be applied to schedplus.

E. TASK MIGRATION OF ROCESS SCHEDULER

To increase CPU utilization, the load balancer migrates tasks running on a relatively busy CPU to a relatively idle CPU. At this time, the process scheduler provides the load balancer with a criterion that the CPU is in an idle if the CPU is in an idle state for a certain period of time. This criterion value is represented by the sched_migration_cost variable [5] of Fig. 3, which is set to a fixed value of 500000 nanoseconds.

If the sched_migration_cost is large, the load balancer selects a CPU that has been in an idle state for a long time and migrates the task running on the busy CPU to the selected idle CPU. This takes a long time to determine if the CPU is in an idle state, but it is accurate. Contrary, if the sched_migration_cost is small, the load balancer can quickly determine if the CPU is in an idle state, but the accuracy will be reduced. If the load balancer incorrectly judges a busy CPU as idle and migrates the task to this CPU, the CPU will be in a very busy state and will have to migrate the task to another idle CPU again. In this case, similar to thrashing, CPU resources are not used for actual task execution, and CPU resources are wasted for task migration.

F. PROCESS SCHEDULER ENHANCEMENT BY POLICY GRADIENT REINFORCEMENT LEARNING

The criterion for determining whether the CPU is idle depends on the characteristics of the running tasks. If most running tasks have short running periods and long sleep periods, the CPU is likely to remain in an idle state even if the CPU is in an idle state for a very short time. Conversely, if most running tasks have long running time and short sleep time, there is a high probability that the CPU will be busy even if the CPU is idle for a while. Therefore, if sched_migration_cost is dynamically changed to match this characteristics of running tasks, process scheduling performance can be improved. To this end, this paper proposes a second function of the Learning EAS that dynamically sets the optimal sched_migration_cost by applying policy gradient reinforcement learning.

When task migration is efficiently performed by the load balancer, workloads per CPU are balanced. In this state, the workload is less concentrated on a specific CPU, and the CPU frequency can be kept low, which reduces power consumption according to the characteristics of Polynomial (7).

State s is the number of task migration attempts, the number of task migration success, and the variance of all CPU workloads in the cluster. a is an action to increase or decrease sched_migration_cost. If the number of task migration attempts, the number of task migration success, and the variance of CPU workloads are large because the number of running tasks is large, sched_migration_cost should be decreased so that task migration is actively performed. Conversely, if the number of task migration attempts, the number of task migration success, and the variance of CPU workloads are small because the number of running tasks is small, sched_migration_cost should be increased to reduce task migration attempts.

When sched_migration_cost is increased or decreased, the policy function must receive a negative reward if the variance of CPU workloads in the cluster increases, and must receive a positive reward if the variance decreases. Parameter θ of the policy function is the policy that increases or decreases sched_migration_cost, this affects the variance of CPU workloads in the cluster. Thus when $\pi(s, a)$ is partially differentiated against θ, it is the rate of the variance change of CPU workloads. However, it is necessary to divide the variance of CPU workloads into the number of task migration attempts and the number of task migration success to calculate the exact reward, because the number of task migration attempts and the number of task migration success increases, the variance of CPU workloads increases.

When the current sched_migration_cost change value is mc_t, the current migration attempts counts is tm_t, the current task migration success counts is ts_t, the current variance of CPU workloads is cv_t, the previous migration attempts counts is tm_{t-1}, the previous task migration success counts is ts_{t-1}, and the previous variance of CPU workloads is cv_{t-1}, for the time t in Equation (1), Equation (11) can be converted into Equation (13).

$$\frac{\partial \rho}{\partial \theta_t} = -mc_t\left(\frac{cv_t}{tm_t + ts_t} - \frac{cv_{t-1}}{tm_{t-1} + ts_{t-1}}\right) \tag{13}$$

When there are m clusters, the Learning EAS implements the policy function as a policy neural network that uses the number of task migration attempts, the number of task migration successes, and the variance of all CPU workloads in each cluster as inputs, and the policy which increases or decreases sched_migraion_cost as an output, as shown in Fig. 6.

Policy neural networks have one hidden layer to minimize computations, and the hidden layer consists of

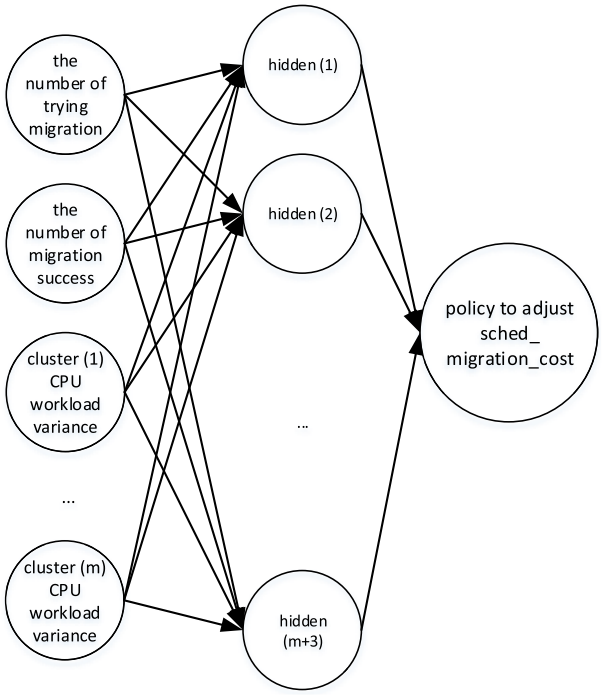

FIGURE 6. Configuration of policy neural network to set sched_migration_cost.

(m + 3) neurons that is one more than the number of input neurons and uses ReLU for activation function.

Algorithm 2 is pseudocode to set sched_migration_cost by running and training policy neural network. The time t is set to 7000 milliseconds so that the policy neural network is executed every 7000 milliseconds.

Each CPU workload in the cluster is stored in one slot every 40ms. Therefore, one slot has a space to store the workload of each CPU in the cluster. CPU workloads in the cluster are stored in 175 slots for 7000ms. The variance of the CPU workloads in each slot is calculated and the average of the calculated 175 variance is set to Algorithm 2's CPU_load_vari. The result of running policy neural network is an output value between 0 and 800000. If the output is greater than the threshold 160000 which is determined by tuning, the policy decreases sched_migration_cost and if it is less, increases sched_migration_cost.

When sched_migration_cost is changed, task migration performance should change meaningfully for the next 7000 milliseconds. This is because the amount of change in task migration performance is used as feedback for policy neural network training. Therefore, Learning EAS changes sched_migration_cost by 17027, which is a prime number over 17000, because sched_migration_cost has a meaningful effect on task migration performance when 17000 or more are changed. The reason for choosing prime number is to minimize the interference between sched_migration_cost value being changed and other variables in CPU scheduler.

The calculation result of Equation (13) is entered as an error so that the policy function is learned through backpropagation and the learning rate α in Equation (1) is adjusted so that the policy neural network stabilizes within 5 minutes.

Algorithm 2 Pseudocode to set Sched_Migration_Cost Using Policy Gradient Reinforcement Learning

```
Initialize policy neural network
CPU_load_vari[m] = {0, 0, ...,0}
prev_total_CPU_load_vari = total_CPU_load_vari = 0
mc = 0
prev_ mig_count = mig_coun = 0
prev_mig_success = mig_success = 0
repeat
  Sleep 7000ms
  total_CPU_load_vari = 0
  i = 0
  while i < m do
    CPU_load_vari[i] = Get the variance of CPU
    workloads in ith cluster
    total_CPU_load_vari + = CPU_load_vari[i]
    + + i
  end
  mig_count = Get task migration attempts counts
  mig_success = Get task migration success counts
  cur_error = (total_CPU_load_vari/
  (mig_count+ mig_success))
  prev_error = (prev_total_CPU_load_vari /
  (prev_mig_count+ prev_mig_success))
  error = −mc ∗(cur_error − prev_error)
  Training policy neural network by back
  propagation(error)
  result = Run policy neural network
  (mig_count, mig_success, CPU_load_variance)
  if (threshold(160000) < result) then
    mc = −17027
  else then
    mc = 17027
  end
  sched_migration_cost + = mc
  prev_total_CPU_load_vari = total_CPU_load_vari
  prev_ mig_count = mig_count
  prev_ success_count = success_count
until system has been shutdown
```

G. IMPLEMENTATION OF LEARNING EAS

Learning EAS is implemented as a library which provides artificial neural network function in Linux kernel, and the policy neural network modules using that library for setting TARGET_LOAD and for setting sched_migration_cost, as shown in Fig. 7.

IV. PERPORMANCE EVALUATION

Learning EAS has been implemented to reduce power consumption and improve process scheduling performance.

TABLE 2. Comparison of train sim current consumption and gaming performance in LG G8 ThinQ.

Game Graphics Menu		EAS			Learning EAS (ms)		
Resolution	Frame Rate	Current Consumption (mA)	Frame Rate (FPS)	Frame Drop Count	Current Consumption (mA)	Frame Rate (FPS)	Frame Drop Count
Low (540 x 960)	Low (30 FPS)	250.91	29.99	1	236.77	29.98	2
Middle (810 x 1440)	Middle (45 FSP)	268.89	44.98	3	254.6	44.99	1
High (1080 x 1920)	High (60 FSP)	331.63	59.91	16	324.15	59.91	17

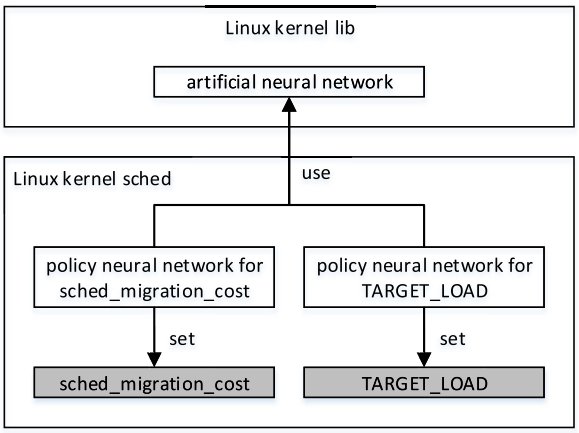

FIGURE 7. Diagram of policy gradient reinforcement learning for learning EAS.

TABLE 3. Current consumption EAS and learning EAS for policy neural network management in LG G8 ThinQ.

Scene	EAS (mA)	Learning EAS without changing TARGET_LOAD and sched_migration_cost (mA)
Music Play	31.48	31.55
Home Idle State	116.12	116.07
Train Sim	331.56	331.43

The power consumption of the Learning EAS and the EAS were measured to compare the power consumption in subsection A, and the hackbench results and entry times of various applications were measured to compare the process scheduling performance in subsection B. CPU scheduling performance of the Learning EAS and EAS were also measured over time to verify the effectiveness of reinforcement learning in subsection C.

A. POWER CONSUMPTION EVALUATION

Power consumption was measured on LG G8 ThinQ [27] using snapdragon 855 chipset platform [28]. The current consumption was measured with the voltage fixed at 4V using the Power Monitor [29] whereas running Train Sim [30] which has various CPU workloads for 3 minutes in the same scenario. When running Train Sim, the current consumption was measured three times by setting both resolution and frame rate to low, normal, and high in the Game graphics menu in the Gaming menu. In addition to measuring the current

TABLE 4. Comparison of train sim current consumption on various chipset platforms.

Model	Chipset Platform	EAS (mA)	Learning EAS (mA)
Q70	Qualcomm Snapdragon 675 [31]	347.04	336.58
K40	Qualcomm Snapdragon 450 [32]	303.67	295.08
Q60	MediaTek Helio P22 [33]	336.76	310.57
K20	MediaTek MT6739 [34]	333.79	316.12

TABLE 5. Comparison of hackbench benchmark results in LG G8 ThinQ.

EAS (s)		Learning EAS (s)
sched_migration_cost = 500000	sched_migration_cost = 10000	
23.5	18.0	17.5

consumption, we also measured the actual train sim's FPS and the number of frame drop to verify the performance change. The reason is that the performance of game application is usually measurable by FPS, and the number of frame drop that failed to generate and display a frame at the target time is a factor of usability.

The average of the three current consumption measurement results are shown in Table 2 and the Learning EAS improved power consumption 2.3% - 5.7% over the EAS. As the current consumption improves and the frame rate and the number of frame drops are not different between the EAS and the Learning EAS, the Learning EAS shows that the power consumption is improved without any performance degradation.

Fig. 8 and Fig. 9 are graphs of current consumption of EAS and Learning EAS measured when running Train Sim with high resolution and high frame rate setting in LG G8 ThinQ. The x axis corresponds to time in minutes and the y axis corresponds to current consumption in mA. It shows that the current consumption changes as CPU workload changes for 3 minutes, and the current consumption of the Learning EAS is kept lower than that of the EAS, as shown in Fig. 10.

The Learning EAS consumes CPU resources to run and training the policy neural network for the TARGET_LOAD adjustment every 100 milliseconds and the policy neural

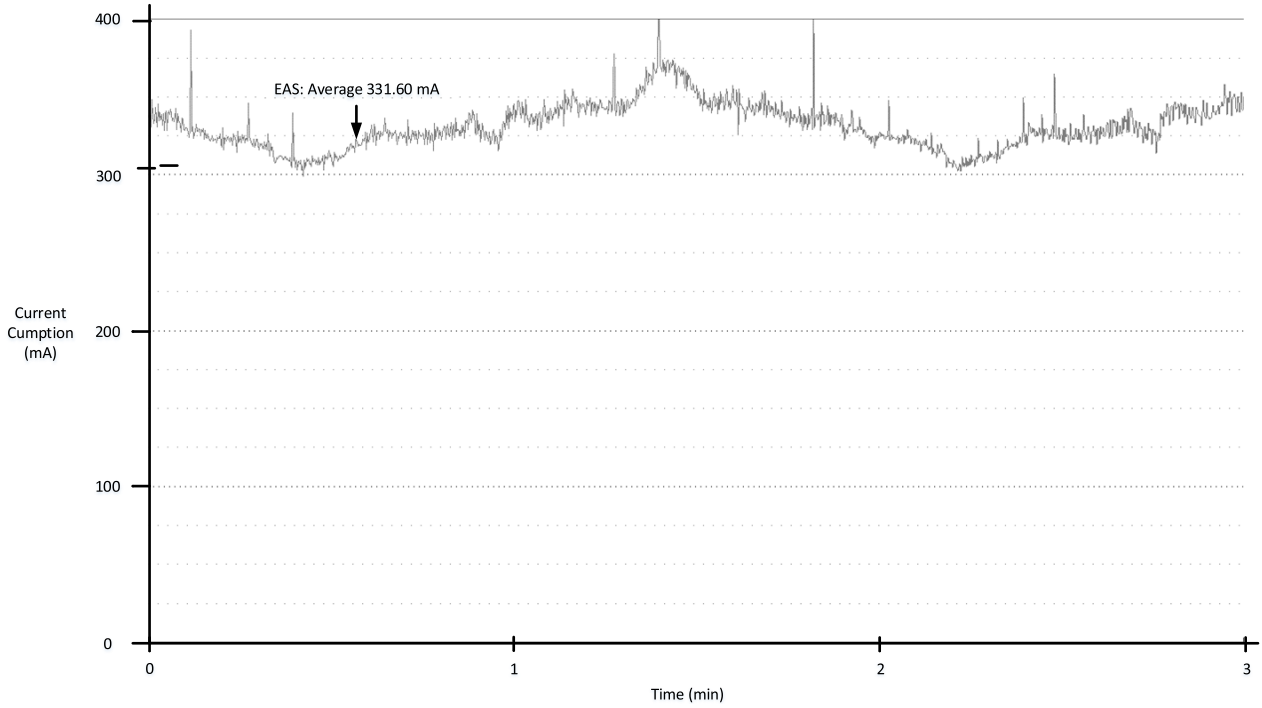

FIGURE 8. Train sim current consumption graph of EAS in LG G8 ThinQ.

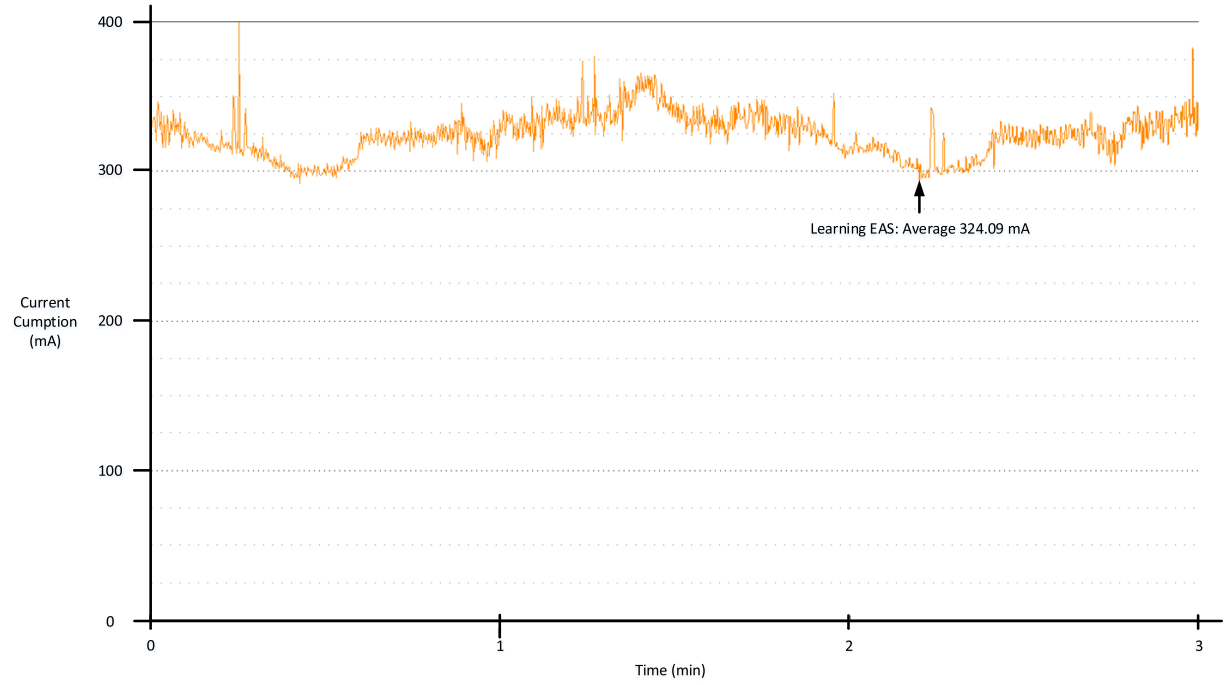

FIGURE 9. Train sim current consumption graph of learning EAS in LG G8 ThinQ.

network for the sched_migration_cost adjustment every 7000 milliseconds. If the power consumption increase for managing policy neural networks is greater than the power reduction obtained through the Learning EAS, the power consumption of the entire system may increase. To measure only the power consumption for managing policy neural networks, the current consumption was measured after the management of policy neural networks were made identical to the Learning EAS without changing TARGET_LOAD and sched_migration_cost.

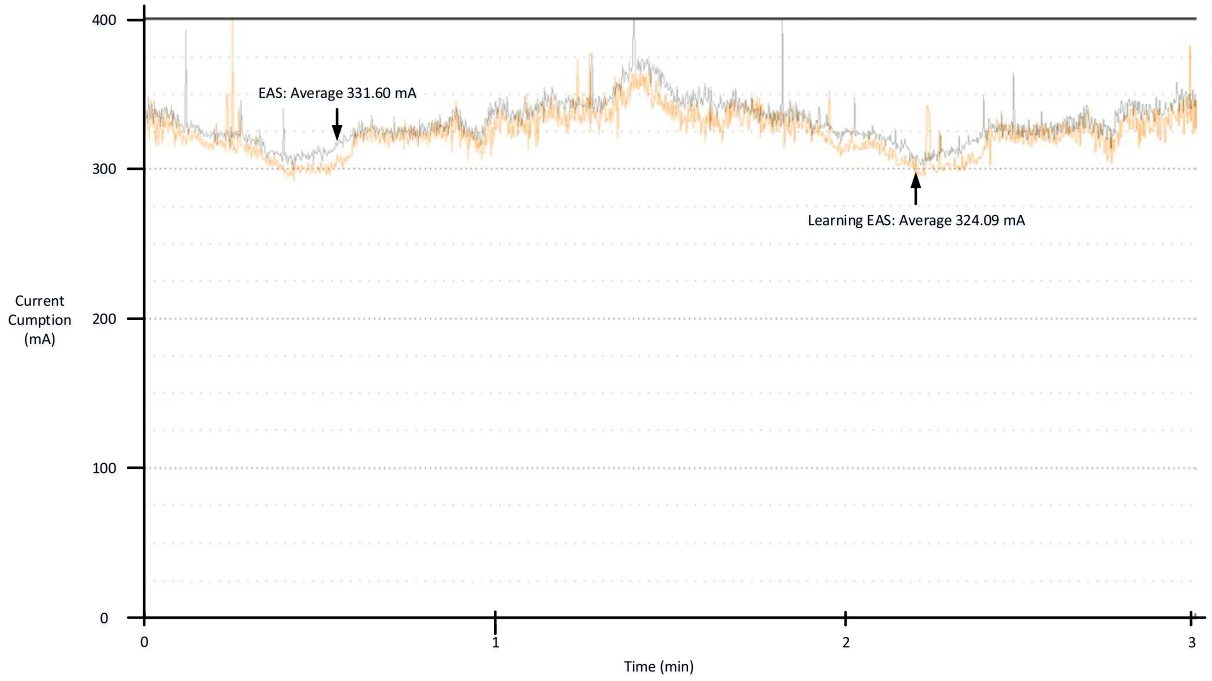

FIGURE 10. Comparison of train sim current consumption graphs of EAS and learning EAS in LG G8 ThinQ.

TABLE 6. Comparison of entry time of applications in LG G8 ThinQ.

Application Name	Category	EAS (ms)		Learning EAS (ms)	Improvement Ratio of Learning EAS (%)	
		(1) sched_migration_cost = 500000	(2) sched_migration_cost = 10000		Compared to (1)	Compared to (2)
Contacts	LG Basic Application	245	258	230	6.1	10.9
Music		162	163	148	8.6	9.2
Message		171	177	157	8.2	11.3
Phone		304	306	291	4.3	4.9
Calculator		267	259	253	5.2	2.3
Camera		325	320	308	5.2	3.8
Gallery		240	240	219	8.8	8.8
Clock		194	203	183	5.7	9.9
Calendar		319	336	300	6.0	10.7
Setting		326	327	314	3.7	4.0
Memo		206	213	191	7.3	10.3
Google	Google Mobile Service (GMS)	130	129	118	9.2	8.5
YouTube		640	626	616	3.8	1.6
PlayStore		355	351	342	3.7	2.6
Gmail		314	307	294	6.4	4.2
Maps		687	660	648	5.7	1.8
Daum	Android Market Application	359	358	349	2.8	2.5
Beach Buggy Blitz		337	328	319	5.3	2.7
Angry Birds		538	542	527	2.0	2.8
Twitter		837	722	705	15.8	2.4
Facebook		294	293	283	3.7	3.4
Average		**345**	**339**	**324**	**6.1**	**4.4**

Current consumption with the voltage fixed at 4V was measured for these three scenes: running music play for 3 minutes with very few CPUs, idle state of home screens for 3 minutes with relatively low CPU usage, and running Train Sim for 3 minutes at high resolution and high frame rate settings with relatively high power consumption.

Table 3 is the average of three measurements of current consumption in the EAS and Learning EAS that do not change TARGET_LOAD and sched_migration_cost.

As a result, the use of CPU resources to manage the policy neural network of the Learning EAS does not affect the increase in power consumption. This is because the size of

TABLE 7. Comparison of entry time of applications with hackbench in LG G8 ThinQ.

Application Name	Category	EAS (ms)		Learning EAS (ms)	Improvement Ratio of Learning EAS (%)	
		(1) sched_migration_cost = 500000	(2) sched_migration_cost = 10000		Compared to (1)	Compared to (2)
Contacts	LG Basic Application	446	421	399	10.5	5.2
Music		333	325	298	10.5	8.3
Message		381	360	343	10.0	4.7
Phone		483	500	463	4.1	7.4
Calculator		467	520	398	14.8	23.5
Camera		525	635	508	3.2	20.0
Gallery		434	428	379	12.7	11.4
Clock		389	410	318	18.3	22.4
Calendar		571	558	466	18.4	16.5
Setting		529	489	464	12.3	5.1
Memo		414	422	377	8.9	10.7
Google	Google Mobile Service (GMS)	556	548	458	17.6	16.4
YouTube		918	888	796	13.3	10.4
PlayStore		795	802	630	20.8	21.4
Gmail		1328	1399	1251	5.8	10.6
Maps		1090	1101	1022	6.2	7.2
Daum	Android Market Application	679	720	661	2.7	8.2
Beach Buggy Blitz		737	759	685	7.1	9.7
Angry Birds		838	866	792	5.5	8.5
Twitter		1211	1428	1107	8.6	22.5
Facebook		812	823	794	2.2	3.5
Average		**664**	**686**	**600**	**9.6**	**12.5**
hackbench		**5517000**	**4935000**	**4726000**	**14.3**	**4.2**

TABLE 8. Comparison of entry time of applications on various chipset platforms.

Model	Chipset Platform	EAS (ms)	Learning EAS (ms)
Q70	Qualcomm Snapdragon 675	624	579
K40	Qualcomm Snapdragon 450	1139	1107
Q60	MediaTek Helio P22	1123	1098
K20	MediaTek MT6739	1799	1715

the actual executing codes to manage policy neural network which confirmed by real-time debugging through Trace 32 is less than 2000 instructions. These instructions can only be processed in 7 microseconds when the CPU is running with a minimum CPU frequency of 300MHz.

We applied the Learning EAS to the high and low specification chipset platforms of both Qualcomm and MediaTek, and measured the power consumption of the Train Sim as described above. Table 4 shows the average of three measurements of Train Sim's current consumption with high resolution and high frame rate on each chipset platform. Learning EAS reduced power consumption by 2.8% -7.8% compared to EAS on various chipset platforms.

B. PROCESS SCHEDULING PERFORMANCE EVALUATION

hackbench [35] is a benchmark program that can measure process scheduling performance. We ran hackbench with the *"hackbench 1 process 100000"* command in android shell and measured the results. *"hackbench"* is the name of the hackbench program you want to run, and the next argument number *"1"* is the number of groups of processes that hackbench creates for testing, with one group consisting of 20 producer processes and 20 consumer processes. The next argument *"process"* then means create producer and consumer tasks as process, not thread. The last argument *"100000"* means that the producer consumer process executes 100000 communications each other.

The average of five hackbench results of EAS with 500000 sched_migration_cost and 10000 sched_migration_cost, and the result of Learning EAS are shown in Table 5.

A smaller heckbench test result indicates a higher process scheduling performance. In general, lowering sched_migration_cost improves hackbench results. Thus, we compare the two results of the EAS with one of the Learning EAS. The two results of the EAS consist of 500000, the default value of sched_migration_cost, and 10000, the value of sched_migration_cost, which shows the best hackbench score. The Learning EAS improves process scheduling performance from 2.8% up to 25.5%, rather than setting the fixed sched_migration_cost in the EAS.

In order to measure the performance improvement effect in a real user scene where the user may notice a difference in performance, we measured entry times of various applications by executing the applications repeatedly. Application execution time quantitatively shows the effect of process scheduling performance on the user because one or more tasks are created when the application is executed and the

TABLE 9. Improved CPU scheduling performance by reinforcement learning in LG G8 ThinQ.

Time (min)	EAS (sched_migration_cost = 500000)				Learning EAS			
	Entry Time (ms)			Current Consumption (mA)	Entry Time (ms)			Current Consumption (mA)
	Contact	Asphalt 9	Chrome		Contact	Asphalt 9	Chrome	
1	249	1177	382	314	246	1157	361	310
2	221	1142	346	302	237	1122	351	292
3	255	1125	358	320	234	1126	350	291
4	223	1113	369	309	223	1121	348	303
5	222	1129	380	316	232	1128	356	294
6	258	1123	364	309	236	1103	350	295
7	245	1117	351	319	235	1123	347	290
8	244	1219	369	318	221	1083	334	288
9	221	1158	374	304	209	1066	335	287
10	252	1121	372	307	213	1053	337	286

same code for each task is executed and the completion time of the code execution becomes the application entry time.

The test method is to execute a single application and measure the entry time and exit the application via the menu button instead of back button. This is repeated 10 times for one application to measure entry times and get the average of measured entry times. This test is run for each of the 21 applications. To improve the accuracy of the test, both the test and the entry time measurement were performed with the script so that the same operation was always performed at the same time.

The entry times for the 21 applications in the LG G8 ThinQ are shown in Table 6. Learning EAS improved the average of application entry times by 6.1% compared to EAS with 500000 sched_migration_cost and by 4.4% with 10000 sched_migration_cost, respectively.

We measured the entry time of the applications when the CPU workload was high due to many running tasks. This test reflects the user scenario when another application is launched whereas one application is already running on the dual display smartphone such as LG V50 ThinQ [36]. We performed the same test for entry times of applications with hackbench running with *"hackbench 1 process 20000000"* command in the background to increase CPU workload.

The results of measuring the entry times of the applications and hackbench are shown in Table 7. The Learning EAS improved the average of application entry times by 9.6%, and 12.5%, respectively, and the hackbench result by 14.3%, and 4.2%, respectively, compared to the EAS with 500000 sched_migration_cost and 10000 sched_migration_cost.

As shown in Table 8, when the Learning EAS was applied to the high and low specification chipset platforms of both Qualcomm and MediaTek chipset vendors, the average of application entry times were reduced by 2.2% - 7.2% compared to the EAS. The reason for the difference in the improvement ratio of applications entry time is that the CPU performance and main memory size for each model are different. In conclusion, the Learning EAS can be used to improve process scheduling performance on various chipset platforms.

C. MEASUREMENT OF CPU SCHEDULING PERFORMANCE GRADUALLY IMPROVED BY REINFORCEMENT LEARNING

Policy function is continuously trained according to the characteristics of running tasks by reinforcement learning. Therefore, the power consumption of Learning EAS is reduced and the process scheduling performance is improved over time by the trained policy function.

To verify this, we repeatedly execute three applications with different characteristics and measured power consumption and application entry time. We execute and terminate the contacts application with a small number of tasks and a small workload of each task, asphalt 9 application with a large number of tasks and a large workload of each task, and chrome web application, in other. It takes 20 seconds to execute and terminate each application, and three applications are executed and terminated in one minute. The current consumption with the voltage fixed at 4V for that one minute is measured.

The results of measuring the entry times of the applications and average current consumption for one minute are shown in Table 9. In the case of EAS, performance does not improve over time, but in case of Learning EAS, applications entry time and current consumption are improved by reinforcement learning.

V. FUTURE WORK

EAS is the latest Linux kernel CPU scheduler that allows software to use big.LITTLE CPUs efficiently. Because EAS is in itself a very large and complex software module, it is difficult to apply machine learning throughout EAS extensively. Therefore, in this paper, we applied the policy gradient reinforcement learning method to only the two variables, TARGET_LOAD which is used to determine the CPU frequency and sched_migration_cost which affects the process scheduling performance greatly.

In order to improve the performance of the Learning EAS and overcome its limitations, it is necessary to improve the

policy gradient reinforcement learning. For this, a study is needed to deeply construct the layer of the policy neural network with a small amount of computation.

Many values that are the basis of operation are determined through heuristics or tuning by the developer or user, including not only the CPU scheduler of the Linux kernel, but also the memory management system or file system. This means that many values can be optimized for a given situation through machine learning. Therefore, as future works, it is necessary to study the performance optimization by applying machine learning to various subsystems of the Linux kernel.

REFERENCES

[1] Gsmarena. (2019). *Gsmarena Phone Finder*. [Online]. Available: https://www.gsmarena.com/search.php3?

[2] B. Jeff. (2013). Big. LITTLE technology moves towards fully heterogeneous global task scheduling. ARM Ltd., Cambridge, UK. [Online]. Available: https://www.arm.com/files/pdf/big_LITTLE_technology_moves_towards_fully_heterogeneous_Global_Task_Scheduling.pdf

[3] ARM Ltd., Cambridge, U.K. (2019). *Energy Aware Scheduling (EAS)*. [Online]. Available: https://developer.arm.com/tools-and-software/open-source-software/linux-kernel/energy-aware-scheduling

[4] MediaTek Inc., Hsinchu, Taiwan. (2015). *Hot Plug Strategy*. [Online]. Available: https://android.googlesource.com/kernel/mediatek/+/android-mtk-3.18/drivers/misc/mediatek/base/power/mt8173/mt_hotplug_strategy_algo.c

[5] I. Molnar, (2007). Completely Fair Scheduling (CFS) class. Linux Foundation, San Francisco, CA, USA. [Online]. Available: https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/kernel/sched/fair.c

[6] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 12, 2000, pp. 1057–1063.

[7] ARM Ltd., Cambridge, U.K. (2019). *Cortex-A15*. [Online]. Available: https://developer.arm.com/ip-products/processors/cortex-a/cortex-a15

[8] ARM Ltd., Cambridge, U.K. (2019). *Cortex-A7*. [Online]. Available: https://developer.arm.com/ip-products/processors/cortex-a/cortex-a7

[9] ARM Ltd., Cambridge, U.K. (2018). *ARM Architecture Reference Manual ARMv7-A and ARMv7-R Edition*. [Online]. Available: https://developer.arm.com/docs/ddi0406/latest/arm-architecture-reference-manual-armv7-a-and-armv7-r-edition

[10] Akashbelekar. *ARM Big.Little is Becoming Trendy*. Tech4Gizmos. Accessed: Jul 24, 2015. [Online]. Available: http://tech4gizmos.com/arm-big-little-is-becoming-trendy

[11] Google Inc., Mountain View, CA, USA. *Energy Model*. [Online]. Available: https://android.googlesource.com/kernel/common.git/+/experimental/android-4.14

[12] Google Inc., Mountain View, CA, USA. *Google Pixel 2*. [Online]. Available: https://www.gsmarena.com/google_pixel_2-8733.php

[13] Qualcomm Inc., San Diego, CA, USA. *Snapdragon 835 Mobile Platform*. [Online]. Available: https://www.qualcomm.com/products/snapdragon/processors/835

[14] ARM Ltd., Cambridge, U.K. (2019). *Cortex-A73*. [Online]. Available: https://developer.arm.com/ip-products/processors/cortex-a/cortex-a73

[15] ARM Ltd., Cambridge, U.K. (2019). *Cortex-A53*. [Online]. Available: https://developer.arm.com/ip-products/processors/cortex-a/cortex-a53

[16] MediaTek Inc., Hsinchu, Taiwan. *MT6750*. [Online]. Available: https://android.googlesource.com/kernel/mediatek/+/android-mtk-3.18/drivers/misc/mediatek/base/power/mt8173/mt_hotplug_strategy_algo.c

[17] D. Brodowski, N. Golde, R. J. Wysocki, and V. Kumar. CPU frequency and voltage scaling code in the Linux(TM) kernel. The Linux Foundation, San Francisco, CA, USA. [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[18] C. Bálint. cpufreq: Interactive: New 'interactive' governor. LWM.net. Accessed: Oct. 27, 2015. [Online] Available: https://lwn.net/Articles/662209/

[19] Y. G. Kim, M. Kim, and S. W. Chung, "Enhancing energy efficiency of multimedia applications in heterogeneous mobile multi-core processors," *IEEE Trans. Comput.*, vol. 66, no. 11, pp. 1878–1889, Nov. 2017.

[20] W. Hu and G. Cao, "Energy-aware CPU frequency scaling for mobile video streaming," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, Jun. 2017, pp. 2314–2321.

[21] A. K. Datta and R. Patel, "CPU scheduling for power/energy management on multicore processors using cache miss and context switch data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 5, pp. 1190–1199, May 2014.

[22] J. Feliu, J. Sahuquillo, S. Petit, and J. Duato, "Perf & fair: A progress-aware scheduler to enhance performance and fairness in SMT multicores," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 905–911, May 2017.

[23] R. Lincoln, S. Galloway, B. Stephen, and G. Burt, "Comparing policy gradient and value function based reinforcement learning methods in simulated electrical power trade," *IEEE Trans. Power Syst.*, vol. 27, no. 1, pp. 373–380, Feb. 2012.

[24] J. Lee, S. Nam, and S. Park, "Energy-efficient control of mobile processors based on long short-term memory," *IEEE Access*, vol. 7, pp. 80552–80560, 2019.

[25] W. Mauerer, *Professional Linux Kernel Architecture*. Birmingham, U.K.: Wrox Press Ltd., 2008.

[26] R. J. Wysocki. *CPUFreq: Schedutil governor*. LWM.net. Accessed: Mar. 4, 2016. [Online]. Available: https://lwn.net/Articles/678777/

[27] LG Electronics Inc., Seoul, South Korea. (2019). *LG G8 ThinQ*. [Online]. Available: https://www.lg.com/us/mobile-phones/g8-thinq

[28] Qualcomm Inc., San Diego, CA, USA. *Snapdragon 855 Mobile Platform*. [Online]. Available: https://www.qualcomm.com/products/snapdragon-855-mobile-platform

[29] Monsoon Solutions Inc., Bellevue, WA, USA. *Power Monitor*. [Online]. Available: https://www.msoon.com/online-store

[30] Train Sim, Calgary, AB, Canada. *3583 Bytes*. [Online]. Available: https://play.google.com/store/apps/details?id=com.ogien.trainsim&hl=en_US

[31] Qualcomm Inc., San Diego, CA, USA. *Snapdragon 675 Mobile Platform*. [Online]. Available: https://www.qualcomm.com/products/snapdragon-675-mobile-platform

[32] Qualcomm Inc., San Diego, CA, USA. *Snapdragon 450 Mobile Platform*. [Online]. Available: https://www.qualcomm.com/products/snapdragon-450-mobile-platform

[33] MediaTek Inc., Hsinchu, Taiwan. *MediaTek Helio P22*. [Online]. Available: https://www.mediatek.com/products/smartphones/mediatek-helio-p22

[34] MediaTek Inc., Hsinchu, Taiwan. *MT6739*. [Online]. Available: https://www.mediatek.com/products/smartphones/mt6739

[35] R. Russell. Hackbench. Canonical Ltd., Boston, MA, USA. Accessed: 2010. [Online]. Available: http://manpages.ubuntu.com/manpages/xenial/man8/hackbench.8.html

[36] LG Electronics Inc., Seoul, South Korea. 2019. *LG V50 ThinQ*. [Online]. Available: https://www.lg.com/us/mobile-phones/v50-thinq-5g

JUNYEONG HAN received the B.S., M.S., and Ph.D. degrees in computer engineering from the School of Computer Science and Engineering, Chung-Ang University, Seoul, South Korea, in 2006, 2008, and 2019, respectively.

He is currently working as a Professional with LG Electronics, Seoul. He has been working to improve performance of the Linux kernel for Android smartphone. His research interest includes Linux systems and machine learning for mobile systems.

SUNGYOUNG LEE received the B.S. and M.S. degrees in computer science from Korea University, Seoul, South Korea, in 1994 and 1996, respectively.

He has been a Research Fellow with LG Electronics, Seoul, since 1996. He holds three domestic patents and one U.S. patents. He is currently developing mobile system software, embedded systems, and embedded Linux systems. His research interest includes Linux memory management and machine learning for mobile memory management optimization depending on user's usage pattern at the mobile.