Hybrid DVFS Scheduling for Real-Time Systems Based on Reinforcement Learning

Fakhruddin Muhammad Mahbub ul Islam and Man Lin

***Abstract*—Power consumption is one of the most challenging issues in the design of modern computing systems. In any computational device, processor consumes significant amount of power compared with other components. In order to reduce power consumption, dynamic voltage and frequency scaling (DVFS) has been commonly used in modern processors. In recent years, there has been much research on real-time DVFS techniques. These techniques work with different strategies and perform well under different conditions. However, a single algorithm is not always optimal under different workloads, dynamic slacks, and power settings. Furthermore, the variation in device configuration also affects the suitability of a given DVFS algorithm. Aiming for adaptability, in this paper, we propose a novel reinforcement learning-based approach, which takes a set of existing techniques, specialized to handle different conditions, and switches to the most suitable one in various situations. Experimental results show that the proposed hybrid approach saves more energy than any single policy executing individually.**

***Index Terms*—Dynamic voltage and frequency scaling (DVFS), energy efficiency, power-aware system, real-time systems, reinforcement learning (RL).**

I. Introduction

POWER consumption problem on portable battery-powered devices has continued to receive a lot of attention in recent years with the higher and higher computational demand and performance requirement under limited battery life. Among various components in a computational device, the processor is one of the major power consumers. Its performance is directly related to its power dissipation and consumes approximately 18%–30% of the total power consumption [1]–[3]. Fig. 1 shows the power consumption breakdown of a Dell XPS M1330 laptop at its maximum settings, which verifies CPU as one of the major power consumers [4]. Power consumption can even exceed 50% for CPU-intensive workloads in newer fabrication technologies [2], [5], [6]. As the demand for performance standards is rapidly increasing, optimizing the processor power consumption has become a major research area.

Power consumption of a silicon-based CMOS processor can be divided mainly into dynamic power and leakage power. Dynamic power P_{dp} is consumed during execution of instructions. It is determined by silicon process technology and is approximately proportional to the clock frequency and square of the operating voltage

$$P_{\mathrm{dp}} \propto V^2 \cdot f.$$

Clock frequency and operating voltage are also approximately proportional in these processors. The study in [7] shows that, with 400-MHz rise in the clock speed, dynamic power consumption increases almost 60%. In contrast, leakage power is mainly determined by silicon processor technology. It is caused by leakage current, which flows in the circuit even while instructions are not being executed [1]. When a silicon device operates at higher clock frequency, dynamic power dominates the total power consumption, and at idle or near-idle condition, leakage power dominates the total power consumption.

Based on the timing constraint, traditionally, real-time systems can be classified into two types, i.e., hard real-time systems and soft real-time systems. Hard real-time systems refer to the systems in which all events must complete their works strictly before deadlines [8]. These systems are often safety critical, and failing to meet the response time requirements can result in a catastrophe, e.g., aircraft control and temperature monitoring system in a nuclear plant. In contrast, the response time requirements of soft real-time systems are not very strict. For example, broadcasting a video can be considered as a soft real-time system.

Dynamic voltage and frequency scaling (DVFS) is a technique in processor architecture where the voltage and frequency can be adjusted at runtime to reduce power dissipation or to decrease the amount of heat produced by the integrated circuit (IC). It achieves a cubic reduction in dynamic power consumption by lowering the supply voltage and operating frequency. However, as the semiconductor fabrication technology keeps scaling down, leakage power becomes more and more dominant

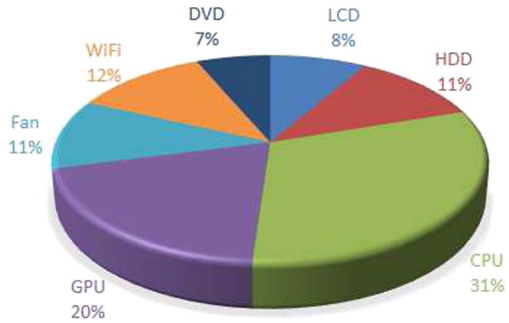

Fig. 1. Power consumption of various components in a Dell XPS M1330 laptop.

Manuscript received November 22, 2014; revised April 2, 2015; accepted May 11, 2015. Date of publication July 21, 2015; date of current version June 26, 2017. This work was supported by the Natural Sciences and Engineering Research Council of Canada.

The authors are with the Department of Mathematics, Statistics and Computer Science, St. Francis Xavier University, Antigonish, NS B2G 2W5, Canada (e-mail: x2012tvi@stfx.ca; mlin@stfx.ca).

Digital Object Identifier 10.1109/JSYST.2015.2446205

in modern processors. Therefore, voltage–frequency scaling decisions should be made carefully by considering processor configurations.

A number of DVFS techniques have been proposed by researchers in recent years to reduce power consumption for real-time systems. These techniques follow different strategies to analyze workload, slack time, etc., and perform well for a specific set of conditions. For example, some techniques start executing tasks with higher clock frequency and gradually reduce it depending on the actual execution time (AET). Some techniques work with reverse strategy, i.e., start aggressively with lower clock frequency to take the advantage of quadratic dependence on operating voltage by the energy consumption and gradually increase the frequency if required. Energy efficiency of these techniques varies with situations such as changes in workload, slack time, and device power setup. There is no single technique that guarantees optimality in all conditions [9]. Therefore, instead of presenting a new DVFS technique, we propose adopting an adaptive approach that chooses a DVFS technique that is most suitable for the current condition by using reinforcement method to learn the DVFS technique arbitration. This scheme takes a set of existing techniques, suitable for different conditions; adapts to the changes of the environment; and offers overall energy savings, which is better than any individual technique.

The main contributions of this paper are summarized as follows.

1) This paper presents an adaptive approach to use multiple DVFS techniques by switching to the most efficient one during execution of tasks depending on the situation.
2) The approach learns the efficiency of each of the techniques on the fly and gradually becomes an expert in determining the appropriate technique in various conditions.
3) This paper differs from previous works (e.g., [9]) in its learning-based ability while considering different DVFS scaling approaches and the feedback from platform used.
4) The proposed approach is capable to work with different scheduling policies and their supported DVFS techniques.

The remainder of this paper is organized as follows. Section II reviews existing research works on related areas. Section III describes real-time task set and power model that is used to evaluate the proposed approach. Section IV describes the motivation of adopting a hybrid DVFS approach with examples. In Section V, we propose the system design of the learning-based hybrid DVFS approach for real-time systems to minimize power consumption. Section VI presents experimental results of the proposed approach, and Section VII concludes this paper with overall summary.

II. Related Works

DVFS is one of the most promising solutions to optimize energy consumption and control temperature. It is supported by most of the modern computing hardware and operating systems. For example, Intel's *SpeedStep* [10] and AMD's *PowerNow*! [11] technologies allow the system to dynamically adjust processor's voltage and frequency settings. In real-time systems, DVFS has two main objectives, i.e., reduce power dissipation and maintain task timeliness behavior [12]. Most of the DVFS algorithms consider satisfaction of time constraints as a *hard* constraint and generally use dynamic slack time, i.e., the difference between AET and worst case execution time (WCET) of tasks to reduce the voltage–frequency and thus cut down the power consumption.

A. Real-Time DVFS Techniques

Various real-time DVFS techniques have been studied in [2], [3], and [13]–[22]. Pillai and Shin [3] presented three DVFS algorithms, namely, static, cycle-conserving (CC), and look-ahead (LA). The static algorithm adjusts the clock frequency using offline parameters such as periods and execution times so that the relative utilization of the task set becomes one. Note that the utilization of a task is the ratio of execution time to period. The AET of a task might change with periods. In order to take the advantage of this phenomenon, the CC algorithm adjusts the utilization dynamically when tasks are scheduled and completed. The approach assumes the worst case for the scheduled task at the start of each new period. Therefore, initially, the clock frequency starts high and gradually decreases when tasks complete their execution. In contrast, the LA algorithm aggressively starts with a lower frequency and tries to defer as much work as possible until the current deadline. As a result, later, the system might be forced to run at a higher frequency if the tasks take WCET to complete execution.

Aydin *et al.* [2] proposed dynamic reclaiming algorithm (DRA), dynamic reclaiming-one task extension (DR-OTE), and aggressive speed adjustment (AGR) algorithm. DRA maintains a data structure called α queue. Whenever a task arrives, its WCET is pushed at $S_{\rm opt}$ frequency in the α queue. $S_{\rm opt}$ can be defined as the frequency level at which the relative utilization of a task set becomes one. DR-OTE is an extension of the DRA. It uses the idea that, if there is only one task in the queue and its remaining time in $S_{\rm opt}$ frequency is less than the next task arrival (NTA) time, then the frequency can be lowered further to use the entire time until the NTA for the executing task. AGR is also an extension of DRA. When there is more than one task in the queue and all tasks are supposed to complete before NTA, then AGR transfers CPU time among these tasks. Aggressive algorithms usually consume less power when slack time is higher due to the convex relationship between dynamic power and clock frequency [2]. However, if leakage power is comparable with dynamic power, then the outcome could be reverse. Instead of saving, these algorithms might cause more energy dissipation because the system will be required to keep alive for a longer period.

Lawitzky *et al.* [23] proposed a DVFS algorithm in which the amount of dynamic slack of a finished task is appended with the budget of next scheduled task and frequency is adjusted accordingly. In order to adjust the increasing leakage power and balancing the dynamic power, Jejurikar *et al.* [24] presented a DVFS scheme in which tasks are executed above a certain processor speed. Lu and Guo [25] proposed a DVFS approach that analyzes utilization of task set and available CPU cores and determines the clock frequency for each of the tasks.

However, most of the existing DVFS policies mainly try to reduce dynamic power consumption and ignore leakage power.

Works in [12] and [26] compare the energy efficiency of different real-time DVFS techniques. Saha and Ravindran [12] experimented timeliness and energy consumption behaviors of DVFS techniques. They found that the energy efficiency of techniques is highly dependent on the relative power consumption of active and idle states of a processor, which, in turn, is dependent on the characteristics of the platform used.

B. Learning-Based DVFS Techniques

System-level power management incorporates learning-based approaches to handle the variability of applications and hardware. A robust power management technique should be able to improve itself in different environments by learning the optimal decision from past experiences. In recent years, there have been some studies in [9] and [27]–[36] on DVFS techniques using different types of learning approaches.

Using supervised learning, Jung and Pedram [33] proposed a power management framework for multiprocessor systems, which analyzes the system performance from input features and determines the optimal voltage–frequency setting by using a precomputed policy table. However, with different workloads and hardware configurations, it is difficult to train a system with a precomputed policy table and training sets that represent the entire environment.

Dhiman and Rosing [36] proposed a policy by using online learning, which uses processor clock frequency levels as different experts, and at runtime, these are selected based on power consumption and performance penalty. Following a similar strategy, Bhatti *et al.* [9] proposed an approach for interplaying of DVFS techniques for real-time systems, which is also based on power consumption and performance penalty. Decisions for choosing the best experts are taken each time when dynamic slack changes. However, the approach does not exactly use previous experiences to decide future decisions, and there are certain DVFS techniques that show better power savings for the time being, but in the long run, they dissipate more energy. Therefore, a greedy choice might do the reverse.

Using reinforcement learning (RL), Shen *et al.* [29] proposed an approach to determine the optimal frequency in a system to manage temperature, performance, and energy consumption simultaneously. Tan *et al.* [31] presented an RL-based approach to execute tasks within a given performance constraint. Without having prior information of the workload, the authors in [27] proposed a power management technique using RL, which dynamically adapts to the environment. However, most of these approaches are mainly applicable in non-real-time scenarios and thus execute tasks without considering deadlines.

C. System-Level Power Measurement

There are several works available that estimate the amount of power consumption by each task or device in a system from application level, e.g., PowerTOP [37], Joulemeter [38], and pTop [39]. Joulemeter estimates the energy usage of hardware or software by measuring the hardware resources (e.g., CPU, memory, and disk) being used and transforming the resource usage to actual power usage. Bircher and John [40] developed power models for six subsystems, i.e., CPU, memory, chipset, I/O, disk, and GPU, using processor performance counters for online measurement of complete system power consumption. Work in [41] observes the power dissipation characteristics while a device is executing particular type of instructions and then combines these observations to estimate the power dissipation for arbitrary programs. Therefore, without using any external power sensing hardware, power consumption of a system can be estimated by using these tools.

III. System Model

A. Real-Time Task Set Model

Real-time tasks are mainly classified as periodic tasks and sporadic tasks. A periodic task is repeatedly executed at regular intervals, whereas sporadic tasks are released at random time instants and have hard timing constraints [8]. In this paper, assumed target tasks are periodic and can be executed without preserving any sequence. A task set composed of n periodic tasks can be defined by (1), where t_i indicates ith individual task in the task set T

$$T = \{t_1(p_1, w_1), t_2(p_2, w_2), \ldots, t_n(p_n, w_n)\} \tag{1}$$

where each task has its predetermined period p_i and WCET w_i. WCET represents the maximum execution time that a task t_i takes to finish at highest possible processor frequency [1], [42]. Therefore, if the processor frequency decreases, the execution time increases and vice versa. The utilization of a task can be defined as the ratio of execution time to period. Hence, total utilization U of a task set T can be calculated as

$$U = \sum_{\forall\, t_i \epsilon T} \frac{w_i}{p_i}. \tag{2}$$

During each period, every task generates a new instance called *job* expressed by t_i^j where j indicates jth job of ith task. Each job has its specific arrival time, execution time, and deadline. Jobs are usually scheduled according to their priorities. Generally, priorities can be assigned statically or dynamically depending on the scheduling policy [8]. We define the hyperperiod (hyp) of a task set T using (3) where LCM represents least common multiple and h is any positive integer indicating hth hyperperiod

$$hyp = h \times LCM(p_1, p_2, \ldots, p_n). \tag{3}$$

B. Power Model

In order to evaluate the energy efficiency of the proposed approach, we adopt processor power model from existing works [1], [2], [42]. Dynamic power (P_{dp}) is the largest energy-consuming part during instruction execution. It is mostly consumed by charging and discharging the capacitance at each CMOS gate's output. If capacitance of c is charged and discharged with operating voltage V, then the charge moves per cycle are $c \cdot V$. Therefore, with a clock frequency f, the charge

TABLE I
SAMPLE TASK SET

Task	Period	WCET
t_1	4	1
t_2	6	2
t_3	12	3

moves per second are $c \cdot V \cdot f$. Since the voltage is V when the charge is delivered, the energy dissipation per second can be stated as

$$P_{\rm dp} = c \cdot V^2 \cdot f. \tag{4}$$

Clock frequency f is primarily dependent on the operating voltage V. It can be expressed by (5) where $V_{\rm th}$ is the threshold voltage and K_6, L_d, and ϵ are constants that are dependent on the processor fabrication technology

$$f = \frac{(V - V_{\rm th})^{\epsilon}}{L_d \cdot K_6}. \tag{5}$$

Leakage power ($P_{\rm lp}$) has become a primary concern for IC designers in deep submicron process technology, like 70 nm and below, because it has increased to 30%–50% of the total IC power consumption [43]. Subthreshold leakage current ($I_{{\rm sub}n}$) and reverse bias junction current (I_j) are mainly responsible for leakage power. It can be expressed as follows:

$$P_{\rm lp} = V \cdot I_{{\rm sub}n} + |V_{\rm bs}| \cdot I_j \tag{6}$$

where $V_{\rm bs}$ is the body bias voltage. If a task has execution time τ_n at voltage V_n and frequency f_n (normalized with respect to execution time at $f_{\max}$, voltage $V_{\max}$, and frequency $f_{\max}$, respectively), its normalized CPU energy consumption can be estimated as follows [36]:

$$E_n = \lambda \cdot V_n^2 \cdot f_n \cdot \tau_n + \rho \cdot V_n \cdot \tau_n \tag{7}$$

where λ and ρ are the percentage contributions of dynamic and leakage power to the total power consumption at maximum v–f setting. In any processor fabrication, λ and ρ can be considered as constants.

IV. MOTIVATIONAL EXAMPLE

In order to describe the requirement of a learning-based hybrid DVFS approach, we demonstrate two simple and well-known techniques: CC and LA. We will show that these two techniques outperform each other in different conditions, and by switching them at appropriate situation, energy consumption can be reduced. Suppose that there is a task set given in Table I with periods and WCETs. Table II shows two sets of AET, i.e., *AET_set-1* and *AET_set-2*. These sets include AET of first 3 jobs for each of the tasks, and t^j represents the jth job of task t. In the first hyperperiod, i.e., within 0 to $1 \times LCM(4, 6, 12) = 12$, *AET_set-1* has dynamic slack of 10% and *AET_set-2* has 45% [calculated using (10)]. We assume that the frequency is continuous and power dissipation function $p(f) = f^3$ [25]. Fig. 2 shows execution of the task set using earliest deadline first (EDF) scheduling with *AET_set-1*.

TABLE II
AET SETS FOR THE TASK SET GIVEN IN TABLE I

	AET_set-1			AET_set-2		
Task	t^1	t^2	t^3	t^1	t^2	t^3
t_1	0.5	1.0	1.0	0.5	0.5	1.0
t_2	2.0	1.5	2.0	2.0	0.5	0.5
t_3	3.0	2.5	3.0	1.0	1.5	1.0

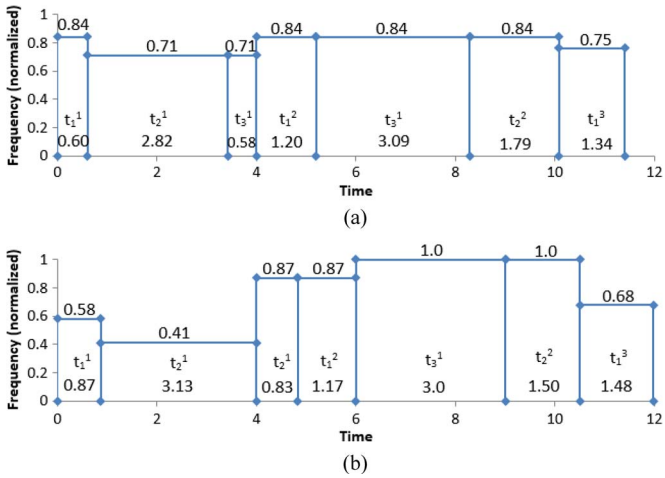

Fig. 2. Execution of the task set given in Table I with *AET_set-1*. (a) CC algorithm. (b) LA algorithm.

Fig. 2(a): According to the CC algorithm, at time $= 0$, the utilization of the task set is $(1/4) + (2/6) + (3/12) \approx 0.84$. Therefore, the clock frequency is set to $f = 0.84$. After the completion of t_1^1 at time $= 0.60$, the utilization is updated with the AET of t_1^1, which is $(0.5/4) + (2/6) + (3/12) \approx 0.71$. With this clock frequency ($f = 0.71$), task t_2^1 is executed up to 3.42. At time $= 3.42$, the utilization is updated as $(0.5/4) + (2/6) + (3/12) \approx 0.71$, and t_3^1 is executed with $f = 0.71$. At time $= 4$, when t_1^2 is released again, the utilization is updated as $(1/4) + (2/6) + (3/12) \approx 0.84$. Therefore, t_1^2 is executed with $f = 0.84$ and t_3^1 is preempted. In this way, at each scheduling event (e.g., task's release and completion), the utilization is updated and frequency is selected accordingly.

Fig. 2(b): At time $= 0$, the LA algorithm determines whether it can finish task t_3^1 after the current deadline 4 and before t_3^1's deadline 12. It is calculated as $U = 0.84 - (3/12) = 0.59$ and $x = \max(0, 3 - (1 - 0.59) \times (12 - 4)) = 0$. $x = 0$ indicates that the whole t_3^1 can be executed within 4 and 12, but U between times 4 and 12 increases to $U = 0.59 + ((3 - x)/(12 - 4)) \approx 0.97$. Next, the LA algorithm calculates whether t_2^1 can be completed within 4 and 6. It is calculated as $U = 0.97 - (2/6) = 0.64$ and $x = \max(0, 2 - (1 - 0.64) \times (6 - 4)) = 1.28$. $x = 1.28$ indicates that part of t_2^1 can be executed within 4 and 6. The rest of t_2^1 will have to be executed within 4. Therefore, the current frequency is calculated as $f = ((0 + 1.28 + 1)/4) \approx 0.58$. At each scheduling event, the technique determines the required frequency and executes the tasks in this way.

Tasks with *AET_set-2* are executed in a similar way. By calculating the energy consumption, we can observe that the CC algorithm consumes approximately 15% less energy than

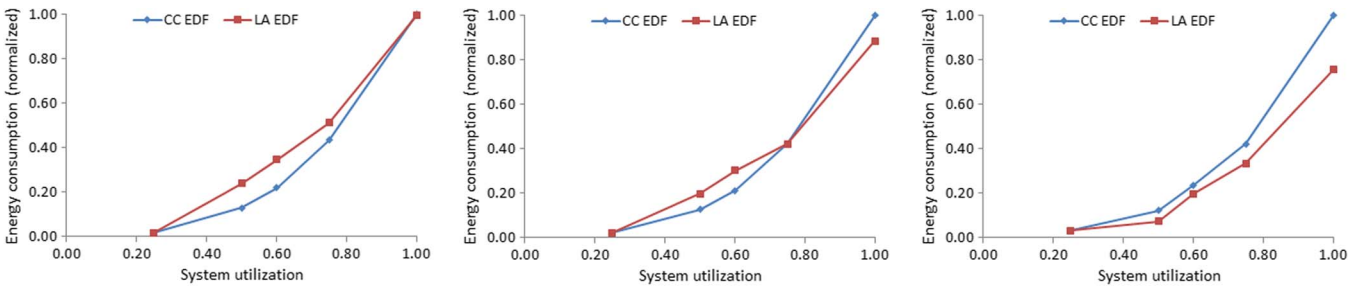

Fig. 3. Energy consumption of CC and LA algorithms with variation in system utilization and dynamic slack. (a) Without slack time. (b) 25% slack time. (c) 50% slack time.

the LA algorithm in *AET_set-1* and the LA algorithm consumes approximately 20% less energy than the CC algorithm in *AET_set-2*. The reason behind this is related to the system utilization and dynamic slack. *AET_set-1* has less slack time, and this is why LA algorithm has been forced later to execute with higher frequency. On the other hand, *AET_set-2* has higher slack time; due to the start with a higher frequency, the CC algorithm consumes more energy in this case.

For motivation purposes, we also experimented the energy consumption of different task sets (consisting of five tasks) by varying their utilizations and dynamic slacks, as shown in Fig. 3. It shows that, with the change of system utilization and dynamic slack, sometimes, the CC algorithm outperforms the LA algorithm, and other times, the LA algorithm outperforms the CC algorithm. It indicates that the combination of these techniques can perform efficiently than using a single policy because the system will be able to switch the DVFS technique with the change of system states. In order to map between the system states and appropriate DVFS policy, a learning-based approach is required because, with various devices and platforms, power consumption of DVFS policies also varies and, hence, state–action maps.

V. Learning-Based Hybrid DVFS

In real-time systems, energy consumption of DVFS algorithms can surpass one another under different workloads, dynamic slacks, device power characteristics, etc. In this paper, we propose an energy-efficient DVFS controlling mechanism by using model-free constrained RL, which handles different DVFS techniques on the fly by executing one of them in appropriate situations. This learning process tries an action in a certain state and adjusts itself when the same state is revisited based on received reward/penalty.

A. RL

RL is a computational approach to learn by interacting with an environment [44]. An RL agent learns from the consequences of its actions, instead of being explicitly taught and determines the mapping between situations and actions. During interaction with the environment, it tries to maximize the total amount of rewards it receives. RL tasks can be classified into two types, i.e., nonsequential and sequential. In nonsequential tasks, an agent learns mapping between situations and actions to maximize its immediate reward. In contrast, in sequential tasks, actions selected by agents may affect its future situations and thus its future rewards. RL is a viable way in many complex domains to train a system.

Q-Learning Algorithm: Q-learning is a commonly used learning approach in RL [27], [45]. In Q-learning, the system consists of a state space S and a set of actions A. Selecting an action $a \in A$ at state $s \in S$ leads the system to a new state with a reward or penalty [29]. A policy π is a mapping from the set of environment states to the set of actions, i.e., $\pi : S \rightarrow A$. For each state–action pair (s, a), it maintains a value function $Q^{\pi}(s, a)$ that represents the penalty or reward. Based on the value function, the agent decides which action should be taken in current state to achieve the maximum rewards. The number of system states and actions must be finite. Iterative update of the Q-value function is the core of the Q-learning algorithm. The Q-value for each state–action pair is updated in the Q-table each time an action is issued and a penalty is received based on the following expression:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(s_t, a_t) \times \left[p_t(s_t, a_t) + \delta \cdot \min_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \quad (8)$$

where $p_t(s_t, a_t)$ is the penalty received in state s_t with action a_t taken. α_t is the learning rate that determines what extent the newly acquired information will override the old information. δ is the discount rate that determines the importance of future rewards and plays an important role when the environment is sequential. $Q(s_{t+1}, a_{t+1})$ is determined based on the action, which costs minimum Q-value. The next time when state s_t is visited again, the action with the minimum Q-value is chosen.

B. System Design

The system design of hybrid DVFS approach based on the Q-learning algorithm is shown in Fig. 4. We assume that the *DVFS controller* consists of N DVFS techniques, which are numbered as $D_1, D_2, \ldots, D_N$. At any scheduling event, i.e., task's release, completion, preemption, or dispatch, *Scheduler* invokes *DVFS controller* to adjust the voltage–frequency

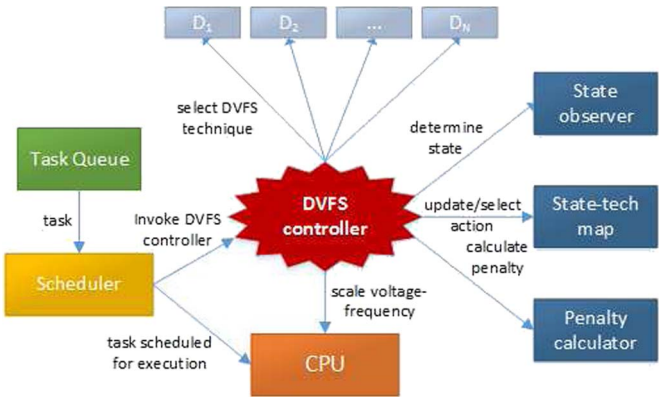

Fig. 4. System design of learning-based hybrid DVFS.

setting. Then, the controller determines whether the time instant is a hyperperiod or not. At a hyperperiod, the controller works as follows:

1) invokes *penalty calculator*, which calculates the energy consumption in the last hyperperiod to determine the penalty incurred by previously selected DVFS technique;
2) updates the *state-tech map* based on the penalty using (8);
3) invokes *state observer* to determine the present system state;
4) using the present state, finds the appropriate DVFS technique from *state-tech map* that has the lowest Q-value;
5) executes the selected method up to the next hyperperiod and sets voltage–frequency settings accordingly.

If the time instant is not a hyperperiod, then the *DVFS controller* simply adjusts the voltage–frequency setting based on the last chosen DVFS method. We choose to use hyperperiod to evaluate a DVFS technique as a switching point between techniques because of the following reasons.

1) All DVFS techniques do not allow switching between techniques at any point during execution, e.g., CC. Hence, hyperperiod is a safe point where DVFS algorithms can be switched without any disorder.
2) In a hyperperiod, all DVFS techniques execute exactly equal number of jobs. Therefore, it is easier to calculate the energy consumption in a fare way for every hyperperiod.
3) If DVFS techniques are switched very frequently, the migration might cause some computational overhead.

1) State Observer: This module determines the state of the system parameters. Describing the state parameters is very crucial for the system because, depending on these state parameters, the system learns how to act in different states. We define the state space S using a vector of two components (su, ds), where su represents system utilization and ds represents dynamic slack of a task set. These parameters are selected because most of the DVFS techniques mainly analyze utilization and dynamic slack to scale voltage and frequency. Parameters are discretized to obtain a finite state space. In order to find the dynamic slack, at first, we calculate the sum of each task's AET in a hyperperiod as follows:

$$Et_{hyp} = \sum_{i=1}^{n} \sum_{j \in hyp} AET\left(t_i^j\right) \tag{9}$$

where $AET(t_i^j)$ means the AET of the jth job of ith task that belongs to the corresponding hyperperiod. In order to get $0 \leq Et_{hyp} \leq 1$, we normalize it in terms of WCET in a hyperperiod and then deduct from 1 to determine the dynamic slack

$$ds = 1 - \frac{Et_{hyp}}{\sum_{i=1}^{n}\left(\frac{hyp}{p_i} \times w_i\right)} \tag{10}$$

where w_i means the WCET of task t_i and hyp/p_i calculates the number of jobs by task t_i in a hyperperiod. System utilization of a set of n tasks can be calculated using (2) given as follows:

$$su = \sum_{i=1}^{n} \frac{w_i}{p_i}. \tag{11}$$

By interacting with the environment, the system learns the best DVFS technique from a set of techniques in a specific state. Actions of our learning-based approach are decisions switching to a DVFS technique in a certain state at a hyperperiod.

2) Penalty Calculator: The main objective of the proposed approach is to achieve maximum energy savings by maintaining all the deadlines. We use average energy consumption in a hyperperiod as a penalty function. Penalty incurred in a hyperperiod can be calculated as follows:

$$p(s, a) = \frac{En(s, a)}{Et_{hyp}} \tag{12}$$

where $En(s, a)$ is the total energy consumption at state s with action a, and Et_{hyp} is the execution time length in the hyperperiod [see (9)].

3) State-Tech Map: State-tech map (i.e., Q-table) holds the mapping between states and DVFS techniques, where each entry stores the penalty (see Table III). The number of states depends on the discretization of the parameters. Large step size reduces the number of states, but smaller step size increases the performance of the system. Each time a feedback is received, the map is updated using the learning rate. For simplicity, it is often sufficient to use a small fixed value (e.g., 0.1) as learning rate. However, for guaranteed convergence, the rate should decrease with time. For fast convergence, we adopt the idea of variable learning rate approach from [27]. In this approach,

TABLE III
STATE-TECH MAP

states	actions			
$s(su, ds)$	a_1	a_2	...	a_N
(1.0, 1.0)	...	...	...	...
...	...	...	...	...
(0.1, 0.1)	...	...	...	...

instead of a fixed learning rate, it depends on the number of visits to the state–action pair (s, a)

$$\alpha = \frac{\alpha'}{\text{visit}(s, a)} \tag{13}$$

where α' is a given constant learning rate and visit(s, a) indicates the number of visits to (s, a) pair. We assume that the environment is nonsequential because the hyperperiods are not affected by each other and action taken in one hyperperiod does not affect the penalty of future hyperperiods. Therefore, we are interested in immediate rewards and consider the discount rate δ as 0.

Algorithm Learning-Based Hybrid DVFS

Variables: time, τ
task set, T
learning rate, α'
number of DVFS techniques, N
State-tech map, $Q[S, A]$
value of a state–action pair, $Q(s, a)$
scheduling event, E
execution time in a hyperperiod, Et_{hyp}

procedure INITIALIZATION
  initialize State-tech map $Q[S, A]$ by 0
  $Et_{hyp} \leftarrow 0$
end procedure
procedure STATE_OBSERVER (T, Et_{hyp})
  system utilization, $su \leftarrow$ calculate using (11)
  dynamic slack, $ds \leftarrow$ calculate using (10)
  return state $s(su, ds)$
end procedure
procedure PENALTY_CALCULATOR (Et_{hyp})
  calculate energy consumption En in last hyperperiod
  penalty $p \leftarrow$ calculate using (12)
  return p
end procedure
procedure DVFS_CONTROLLER (τ, E)
  if τ is a hyperperiod **then**
    penalty, $p \leftarrow$ Penalty_calculator (Et_{hyp})
    state, $s \leftarrow$ State_observer (T, Et_{hyp})
    variable learning rate, $\alpha \leftarrow$ calculate using (13)
    update $Q(s, a)$ using (8)
    action, $a \leftarrow a_i$ where $i \in [1, N]$ with minimum $Q(s, a)$ value (action for next hyperperiod)
    $Et_{hyp} \leftarrow 0$
  end if
  DVFS technique, $D \leftarrow$ action, a
  set voltage–frequency using D based on event E
  update Et_{hyp} (using AET of executed task)
end procedure

C. Complexity Analysis

1) Time Complexity: We assume that the real-time task set consists of n tasks. Efficient implementation of procedure *State_observer* costs $O(1)$ time complexity because Et_{hyp} is updated at every scheduling event. In the cases where tasks are added or removed at runtime, the worst case time complexity is linear. Procedure *Penalty_calculator* calculates penalty and updates *State-tech map*, which takes constant time. Computational complexity of procedure *DVFS_controller* is mainly dependent on DVFS techniques used in the DVFS set. As we assume that the environment is nonsequential, hence, the time complexity of the Q-learning algorithm to converge to the best solution is not high and depends on the number of states and actions.

2) Space Complexity: Space complexity of the algorithm depends on the discretization of system utilization (su), dynamic slack (ds), and the number of DVFS techniques used in the hybrid set. Suppose that there are N DVFS techniques and $|su| \times |ds|$ system states. Therefore, space complexity of the algorithm is $O(|su| \times |ds| \times N)$. If the space complexity to store Q-table becomes too high, then a neural network can be used to approximate the Q-value at the cost of some computational overhead [28]. In order to maintain variable learning rate, same size of space is also required to store the number of visits.

VI. Experimental Result

In order to evaluate the performance of the proposed approach, we have developed a simulator using C++, which is capable of scheduling real-time tasks with DVFS. It can configure the system to different power settings. The simulator takes a task set, consisting of periods, WCETs, AETs, etc., as input and gives scheduling details and energy consumption as output. DVFS techniques in the hybrid DVFS set are implemented as separate *class* files; hence, any new DVFS technique can be easily integrated with the set. Each *class* implements a common set of procedures indicating different scheduling events. Therefore, the controller can easily call them at a particular event.

In the experiment, randomly generated task sets are scheduled by the simulator, which is a common validation methodology in many of the previous works [1], [3], [9], [24], [42]. In a task set, each task has uniform probability of having small (5–25 ms), medium (26–75 ms), or long (76–120 ms) periods [3], [9]. Using a uniform distribution, AETs are generated within one of the predefined ranges, i.e., 0.25 ± 0.2, 0.5 ± 0.2, and 0.75 ± 0.2 [1]. Therefore, dynamic slack varies during the execution at different hyperperiods. Task sets are executed up to 50 hyperperiods, and each algorithm executes exactly equal number of jobs in this duration. The power model described in Section III-B is used to configure the power setting. We assume that the frequency of the processor is continuous and the lowest possible frequency is $f_{\min} = 0.25$. System utilization su and dynamic slack ds are discretized with a step size of 0.10. Energy consumption is normalized with respect to the highest consumption by any method.

Based on the chosen DVFS techniques in the hybrid DVFS set, the system works as hard or soft real-time systems. In the experiments, we have selected three hard real-time DVFS techniques, i.e., CC, LA, and DRA. As a scheduler, we have selected EDF, which is one of the most renowned dynamic priority scheduling algorithms. According to EDF, runnable job

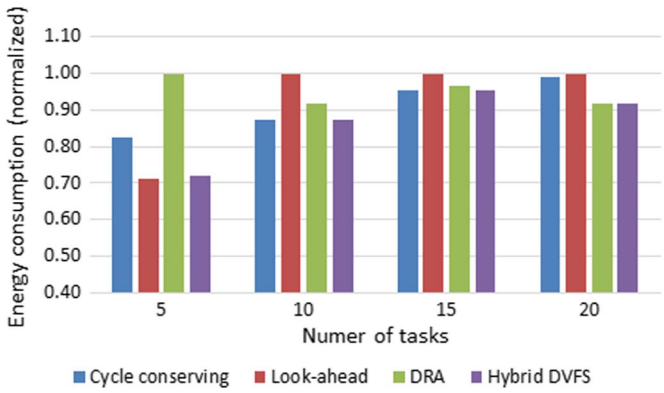

Fig. 5. Variation in the number of tasks.

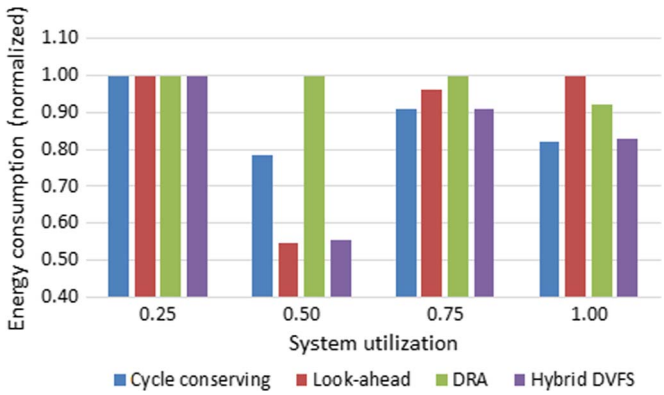

Fig. 6. Variation in system utilization.

with the earliest deadline is always given the highest priority. We have used the learning rate $\alpha' = 0.30$.

A. Varying Number of Tasks

Fig. 5 shows energy consumption among DVFS techniques with variation in the number of tasks. We experimented with 5, 10, 15, and 20 tasks. Each of the task sets has system utilization of 0.60. WCETs in each task set are randomly chosen in the range [0.5 ms, 10 ms]. AET is randomly chosen within one of the predefined ranges. After generation of the task sets, we determine that the task set with five tasks has average dynamic slack of 56%, the 10-task set has 44%, the 15-task set has 44%, and the 20-task set has 48%. We have only considered dynamic power in this experiment.

In Fig. 5, we can observe that, with the five-task experiment, aggressive algorithm LA performs better, and in the rest of the cases, the CC algorithm and DRA perform better. The reason is related to dynamic slack. The task set with five tasks has the highest dynamic slack of 56%. In all of the cases, we can see that hybrid DVFS performs like the best available technique.

B. Varying System Utilization

Fig. 6 shows the energy consumption of different DVFS techniques with variation in system utilization. Various task sets are generated for each of the experiments, and system utilizations vary between 0.25 and 1.0. Each of the task sets consists of ten periodic tasks. WCETs in each task set are randomly chosen in the range [0.5 ms, 10 ms]. AET is randomly chosen within one of the predefined ranges. We determined that the average dynamic slack is within the range [40%–60%]. Only dynamic power is considered in this experiment.

In Fig. 6, we can observe that, with utilization of 0.25, all techniques perform exactly the same. This is because, due to low utilization, all of the techniques run with lowest possible processor frequency. In other cases, techniques have outperformed one another with change of utilization and slack time. However, in any cases, hybrid DVFS is the closest one with the best available policy.

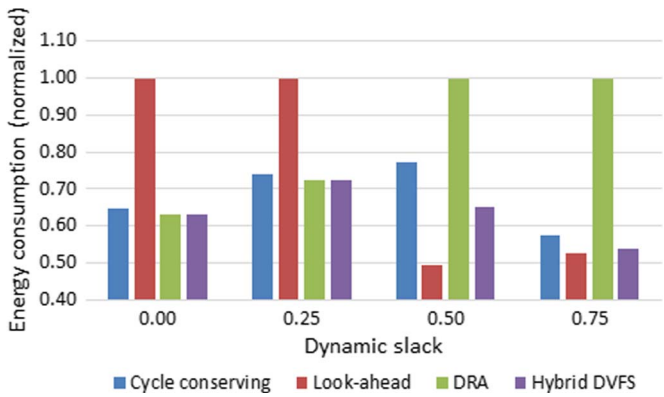

Fig. 7. Variation in dynamic slack.

C. Varying Dynamic Slack

Fig. 7 shows the energy consumption of different DVFS techniques with variation in dynamic slack. Various task sets are generated for each of the experiments with system utilization of 0.50. Each of the sets consists of ten periodic tasks. WCETs in each task set are randomly chosen in the range [0.5 ms, 10 ms]. We have only considered dynamic power in this experiment.

In Fig. 7, we can observe that one DVFS technique might outperform or underperform another with the change of dynamic slack. We have seen previously that the LA algorithm starts with lower clock frequency. Therefore, when the task set has less slack time, the algorithm is forced later to execute with higher frequency, which leads toward more energy consumption. In all of the cases, hybrid DVFS approach follows the best policy. Sometimes, it consumes slightly more than the best technique because, initially, techniques are frequently switched due to the learning process, and dynamic change of situation causes few moments to switch to the best technique.

D. Varying BCET/WCET Ratio

Fig. 8 shows energy consumption of different DVFS techniques with variation in BCET/WCET ratio, where BCET represents best case execution time. The generated task set has utilization of 0.60. Each of the sets consists of ten periodic tasks. AETs of the tasks are randomly chosen in the range [BCET–WCET]. Only dynamic power is considered in this experiment.

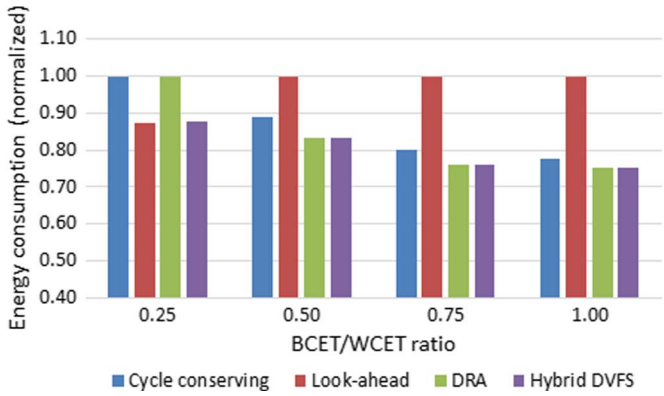

Fig. 8. Variation in BCET/WCET ratio.

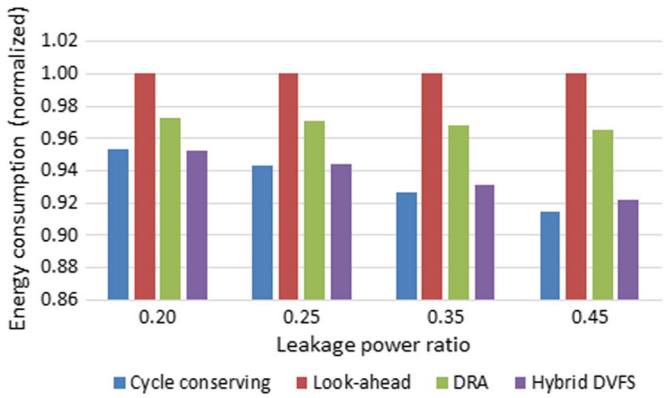

Fig. 9. Variation in leakage power ratio.

In Fig. 8, we can observe that, with BCET/WCET ratio being 1, the system does not have any slack time, and the CC algorithm and DRA perform better. As the BCET/WCET ratio decreases, the LA algorithm starts performing better. The reason is related to the strategies used in these techniques. In any of the cases, hybrid DVFS performs like the best available technique.

E. Varying Leakage Power Ratio

Fig. 9 shows the energy consumption of different DVFS techniques with variation in device leakage power ratio. We have used the same task set with different leakage power ratios for this experiment. The task set consists of ten tasks and has utilization of 0.60. The task set has average dynamic slack of 48%. The power model described in Section III-B has a leakage power ratio of 0.35. Using (7), we see the effect in energy consumption by changing the ratio of leakage power to 0.20, 0.25, 0.35, and 0.45.

In Fig. 9, we can observe that, in all of the cases, the CC algorithm consumes less energy. As the leakage power ratio increases, power consumption difference with the LA algorithm starts increasing. This is mainly because the LA algorithm starts with a lower frequency, and thus, it tends to take longer time to finish the tasks. When the leakage power ratio increases, it leads to more energy consumption for the LA algorithm. The figure also indicates that, with the variation in device power settings, power consumption of DVFS algorithms varies. However, we can see that, with any experiment, hybrid DVFS consumes energy similar to the best performing DVFS technique. Table IV shows the summary of parameters used in different experiments.

TABLE IV
PARAMETERS USED IN DIFFERENT EXPERIMENTS

Parameters/Exp.	A	B	C	D	E
Number of tasks	5-20	10	10	10	10
System utilization	0.60	0.25-1.00	0.50	0.60	0.60
Dynamic slack	40-60%	40-60%	0-75%	40-60%	48%
Leakage power ratio	-	-	-	-	0.20-0.45

VII. CONCLUSION

In this paper, instead of developing a new real-time DVFS technique, we have attempted to take the advantage of different strategies used by each of the existing DVFS techniques. We have tried to make it flexible by using an adaptive learning-based method; hence, with various hardware configurations, power setups, workloads, and other parameters, the proposed approach can discover DVFS policy mapping by itself based on the system state and choose the suitable DVFS technique. Computationally, the algorithm is lightweight, and it has tolerable runtime overhead. Various DVFS techniques can be easily incorporated into our scheme. The approach could be more effective for multicore processors because different cores can run different DVFS techniques based on their system states. Experimental results show that the proposed approach works more efficiently than any individual DVFS technique. Execution characteristic of tasks affects the performance of voltage and frequency scaling. As a future work, we want to take task execution characteristics into consideration and extend the hybrid DVFS approach to multi-core processors.

REFERENCES

[1] E. Seo, J. Jeong, S. Park, and J. Lee, "Energy efficient scheduling of real-time tasks on multicore processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 11, pp. 1540–1552, Nov. 2008.
[2] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Trans. Comput.*, vol. 53, no. 5, pp. 584–600, May 2004.
[3] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 89–102, Oct. 2001.
[4] J. Atwood, "Revisiting "how much power does my laptop really use"?" Apr. 2008.
[5] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "Ecosystem: Managing energy as a first class operating system resource," *SIGARCH Comput. Archit. News*, vol. 30, no. 5, pp. 123–132, Oct. 2002.
[6] J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic voltage scaling on a low-power microprocessor," in *Proc. 7th Annu. Int. Conf. MobiCom*, 2001, pp. 251–259.
[7] W. Knight, "Two heads are better than one [dual-core processors]," *Inst. Elect. Eng. Rev.*, vol. 51, no. 9, pp. 32–35, Sep. 2005.
[8] J. W. S. Liu, *Real-Time Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, 2000.
[9] M. Bhatti, C. Belleudy, and M. Auguin, "Hybrid power management in real time embedded systems: An interplay of DVFS and DPM techniques," *Real-Time Syst.*, vol. 47, no. 2, pp. 143–162, Mar. 2011.
[10] "Enhanced Intel SpeedStep technology for the Intel Pentium M processor," Intel, Santa Clara, CA, USA, Tech. Rep., Mar. 2004.
[11] "AMD PowerNow! Technology," Adv. Micro Devices (AMD) Inc., Sunnyvale, CA, USA, Tech. Rep., Nov. 2011.

[12] S. Saha and B. Ravindran, "An experimental evaluation of real-time DVFS scheduling algorithms," in *Proc. 5th Annu. Int. SYSTOR*, 2012, pp. 7:1–7:12.
[13] L. Yang, M. Lin, and L. T. Yang, "Multi-core fixed priority DVS scheduling," in *Proc. 12th Int. Conf. Algorithms Archit. Parallel Process.*, Y. Xiang *et al.*, Eds., 2012, vol. 7439, pp. 517–530.
[14] L. Yang, M. Lin, and L. Yang, "Integrating preemption threshold to fixed priority DVS scheduling algorithms," in *Proc. 15th IEEE Int. Conf. Embedded RTCSA*, Aug. 2009, pp. 165–171.
[15] E. Bini, G. Buttazzo, and G. Lipari, "Minimizing CPU energy in real-time systems with discrete speed management," *ACM Trans. Embedded Comput. Syst.*, vol. 8, no. 4, pp. 31:1–31:23, Jul. 2009.
[16] F. Dabiri, A. Vahdatpour, M. Potkonjak, and M. Sarrafzadeh, "Energy minimization for real-time systems with non-convex and discrete operation modes," in *Proc. DATE Conf. Exhib.*, Apr. 2009, pp. 1416–1421.
[17] S. Liu, Q. Wu, and Q. Qiu, "An adaptive scheduling and voltage/frequency selection algorithm for real-time energy harvesting systems," in *Proc. 46th Annu. DAC*, 2009, pp. 782–787.
[18] J. Zhuo and C. Chakrabarti, "Energy-efficient dynamic task scheduling algorithms for DVS systems," *ACM Trans. Embedded. Comput. Syst.*, vol. 7, no. 2, pp. 17:1–17:25, Jan. 2008.
[19] P. Juang, Q. Wu, L.-S. Peh, M. Martonosi, and D. Clark, "Coordinated, distributed, formal energy management of chip multiprocessors," in *Proc. ISLPED*, Aug. 2005, pp. 127–130.
[20] W. Kim, J. Kim, and S.-L. Min, "Dynamic voltage scaling algorithm for fixed-priority real-time systems using work–demand analysis," in *Proc. ISLPED*, Aug. 2003, pp. 396–401.
[21] W. Kim, J. Kim, and S. Min, "A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis," in *Proc. Conf. DATE*, 2002, pp. 788–794.
[22] Y. Shin, K. Choi, and T. Sakurai, "Power optimization of real-time embedded systems on variable speed processors," in *Proc. IEEE/ACM ICCAD*, Nov. 2000, pp. 365–368.
[23] M. P. Lawitzky, D. C. Snowdon, and S. M. Petters, "Integrating real time and power management in a real system," in *Proc. Workshop Oper. Syst. Platforms Embedded Real-Time Appl.*, 2008, pp. 1–10.
[24] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proc. 41st Annu. DAC*, 2004, pp. 275–280.
[25] J. Lu and Y. Guo, "Energy-aware fixed-priority multi-core scheduling for real-time systems," in *Proc. IEEE 17th Int. Conf. Embedded RTCSA*, Aug. 2011, vol. 1, pp. 277–281.
[26] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min, "Performance comparison of dynamic voltage scaling algorithms for hard real-time systems," in *Proc. 8th IEEE Real-Time Embedded Technol. Appl. Symp.*, 2002, pp. 219–228.
[27] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 2, pp. 24:1–24:32, Apr. 2013.
[28] R. Ye and Q. Xu, "Learning-based power management for multi-core processors via idle period manipulation," in *Proc. 17th ASP-DAC*, Jan. 2012, pp. 115–120.
[29] H. Shen, J. Lu, and Q. Qiu, "Learning based DVFS for simultaneous temperature, performance and energy management," in *Proc. 13th ISQED*, Mar. 2012, pp. 747–754.
[30] R. Glaubius, T. Tidwell, C. D. Gill, and W. D. Smart, "Real-time scheduling via reinforcement learning," 2012, to be published. [Online]. Available: http://arxiv.org/abs/1203.3481
[31] Y. Tan, W. Liu, and Q. Qiu, "Adaptive power management using reinforcement learning," in *Proc. IEEE/ACM ICCAD*, Nov. 2009, pp. 461–467.
[32] C. T. Liu and R. C. Hsu, "Adaptive power management based on reinforcement learning for embedded system," in *New Frontiers in Applied Artificial Intelligence*, vol. 5027, N. Nguyen, L. Borzemski, A. Grzech, and M. Ali, Eds. Berlin, Germany: Springer-Verlag, 2008.
[33] H. Jung and M. Pedram, "Supervised learning based power management for multicore processors," *Trans. Comput.-Aided Des. Integr. Circuit. Syst.*, vol. 29, no. 9, pp. 1395–1408, Sep. 2010.
[34] N. AbouGhazaleh *et al.*, "Integrated CPU and L2 cache voltage scaling using machine learning," in *Proc. ACM SIGPLAN/SIGBED Conf. LCTES*, 2007, pp. 41–50.
[35] A. Weissel and F. Bellosa, "Self-learning hard disk power management for mobile devices," in *Proc. 2nd IWSSPS*, Seoul, Korea, Oct. 2006, pp. 33–40.
[36] G. Dhiman and T. Rosing, "System-level power management using online learning," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 5, pp. 676–689, May 2009.
[37] "PowerTOP," Intel Open Source Technology Center, Santa Clara, CA, USA. [Online]. Available: https://01.org/powertop
[38] R. Mittal, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," in *Proc. 18th Annu. Int. Conf. Mobicom*, 2012, pp. 317–328.
[39] T. Do, S. Rawshdeh, and W. Shi, "pTop: A process-level power profiling tool," in *Proc. 2nd HotPower*, 2009, pp. 1–5.
[40] W. Bircher and L. John, "Complete system power estimation using processor performance events," *IEEE Trans. Comput.*, vol. 61, no. 4, pp. 563–577, Apr. 2012.
[41] C. Krintz, Y. Wen, and R. Wolski, "Predicting Program Power Consumption," Dept. Comput. Sci., Univ. California, Santa Barbara, CA, USA, Tech. Rep., 2002.
[42] X. Wu, Y. Lin, J.-J. Han, and J.-L. Gaudiot, "Energy-efficient scheduling of real-time periodic tasks in multicore systems," in *Netw. Parallel Comput.*, vol. 6289, ser. Lecture Notes in Computer Science, C. Ding, Z. Shao, and R. Zheng, Eds. Berlin, Germany: Springer-Verlag, 2010.
[43] L. Y. Lu, K. Y. Chen, and T. Y. Wu, "High performance and low leakage design using cell replacement and hybrid V_t standard cell libraries," in *Proc. Int. MultiConf. Eng. Comput. Sci.*, 2008, vol. 1, pp. 248–252.
[44] F. Woergoetter and B. Porr, "Reinforcement learning," vol. 3, no. 3, p. 1448, 2008, revision # 91704.
[45] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Don Mills, ON, USA: Pearson, 2003.

Fakhruddin Muhammad Mahbub ul Islam received the B.Sc. degree in computer science and engineering from Military Institute of Science and Technology (MIST), Dhaka, Bangladesh, in 2009. He is currently working toward the M.Sc. degree in computer science at St. Francis Xavier University, Antigonish, NS, Canada.

From 2010 to 2012, he was a Lecturer with the Department of Computer Science and Engineering, MIST. His current research interests include real-time systems, power-aware scheduling, machine learning, and image processing. His research is supported by the Natural Sciences and Engineering Research Council of Canada.

Man Lin received the B.E. degree in computer science and technology from Tsinghua University, Beijing, China, in 1994 and the Lic. and Ph.D. degrees from Linköping University, Linköping, Sweden, in 1997 and 2000, respectively.

She is currently a Professor of computer science with St. Francis Xavier University, Antigonish, NS, Canada. Her research interests include real-time and embedded system scheduling, power-aware computing, parallel algorithms, and optimization algorithms. Her research is supported by Natural Sciences and Engineering Research Council of Canada.