State-Aware Dynamic Frequency Selection Scheme for Energy-Harvesting Real-Time Systems

Jing Chen, Tongquan Wei, *Member, IEEE*, and Jianlin Liang

***Abstract*— With the increasing deployment of battery-powered embedded systems such as sensor nodes in extreme environments, harvesting renewable energy from ambient environments to achieve near perpetual operation of a system has attracted considerable research efforts in the recent past. In this paper, the authors propose a dynamic frequency selection scheme for energy-harvesting real-time systems. The proposed scheme characterizes the state of a system from the perspectives of system utilization and harvested energy with respect to a certain period of time. A portion of the battery energy is allocated to a group of tasks in the period of time by jointly considering the system utilization and energy state, and the operating frequency is selected based on the allocated energy. The derived operating frequency is fine tuned to further enhance energy efficiency when overflow occurs. Simulation results demonstrate the effectiveness of the proposed scheme. Compared with the state-of-the-art scheme that decouples the energy and timing design constraints, the proposed scheme achieves comparable deadline miss rate when the battery capacity is lower than 5000 J and achieves about 11.5% lower deadline miss rate when the battery capacity is greater than 30 000 J. The proposed scheme also outperforms the benchmarking scheme in energy efficiency. When the battery is near a full charge or overflow occurs, the proposed scheme incurs less energy waste when compared with the benchmarking algorithm, which is favorable for autonomous operation of the system. Furthermore, the time complexity of the proposed scheme is one order of magnitude lower than that of the benchmarking scheme, which makes the proposed scheme well suited for dynamic scheduling.**

***Index Terms*— Dynamic scheduling, real-time systems, state-aware frequency selection.**

I. Introduction

POWER and energy management has become a critical issue in designing real-time embedded systems, especially for battery-powered systems that operate in harsh environments [1]. These systems are generally deployed in extreme environments where human beings have no access and replacing a battery is not practical. Hence, it is desirable that the deployed system itself can harvest energy from the ambient environment to sustain its perpetual operation. Energy harvesting is essentially the conversion of renewable generation into electricity to power electronic devices. Of renewable energy generations, harvesting energy with photovoltaic (PV) array is one of the most popular techniques applied to extract solar energy from the ambient environment.

Extensive research has been conducted in two aspects to improve the efficiency of scavenging and utilizing solar energy. The first aspect of research focuses on obtaining the maximum power output from a PV array. Various PV array maximum power point tracking (MPPT) techniques were summarized and compared in [2]. It has been shown that the two-stage IncCond [3], [4] and the current sweep [5] MPPT methods are appropriate for residential areas, which can be further combined with the irradiance forecasting scheme presented in [6] to estimate the energy output.

The second aspect of the research investigates the system level methods to efficiently exploit the fluctuating energy produced by the solar panel. In [7], the authors prototyped the use of solar energy to power wireless sensor networks. Environment-aware task-scheduling methods were presented to help improve the energy efficiency in distributed system with communication constraints. In [8], throughput optimal and mean delay optimal energy management policies are identified for a sensor node with an energy-harvesting source. These policies can make the system work in energy neutral operation. In [9], the authors presented a feasibility test for the fixed priority and earliest deadline first (EDF) scheduling policies under energy constraints. With the feasibility test, a scheduling scheme was proposed with considerations of the battery capacity and the context switch between the battery recharging mode and discharging model. The design and implementation of a solar-powered system called Prometheus was presented in [10]. The system can operate for an extremely long duration for the wireless sensor network mote-Telos. A similar design was presented in [11] that prototypes a sensor node called Heliomote with energy-harvesting capability. A lazy scheduling algorithm (LSA) was presented in [12] that postpones the execution of a task as late as possible. The optimal start time of the task is derived under the assumption that the task executes at the constant processor speed until its deadline. In [13] and [14], the authors presented a set of algorithms and methods to optimize system performance subject to given energy constraints. The target application scenarios include real-time scheduling, application rate control as well as reward maximization. Both continuous parameters (such as the instantiation rates of tasks and amount of computation) and discrete levels of services are considered

Manuscript received November 6, 2012; revised March 21, 2013; accepted July 25, 2013. Date of publication September 4, 2013; date of current version July 22, 2014. This work was supported in part by the Natural Science Foundation of Shanghai City under Grant 12ZR1409200 and by the Scientific Research Foundation for Returned Scholars, Ministry of Education of China, under Grant 44420340. *(Corresponding author: T. Wei.)*

J. Chen and T. Wei are with the Computer Science and Technology Department, East China Normal University, Shanghai 200241, China (e-mail: 51101201009@ecnu.cn; tqwei@cs.ecnu.edu.cn).

J. Liang was with Broadcom Corporation, San Diego, CA 92617 USA (e-mail: jianlinl@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2013.2278315

to maximize the utility of energy-harvesting systems. These methods optimize system performance that allows smaller solar cells and smaller batteries; however, energy savings are not considered in these designs.

Dynamic voltage scaling (DVS) technique has been widely utilized to save energy in energy-harvesting systems. In [15], the authors proposed an EDF-based real-time scheduling algorithm for uniprocessor energy-harvesting systems that jointly considers the constraints from the energy domain and from the time domain. However, the algorithm schedules tasks at a fixed frequency and does not employs DVS technique for energy management.

The LSA presented in [16] is extended in [17] to enable energy savings by using DVS. When a task arrives, it is pushed into the ready queue, and the runtime schedule is updated for all tasks in the ready queue in two major steps.

1) The lowest possible processor speed is found for all tasks in the ready queue.
2) The energy availability of the system is checked. If the available energy is not enough, the execution of the current task is delayed such that the task has enough energy to finish its execution, and the schedule of all remaining tasks is verified. Otherwise if enough energy is available, the execution of tasks follows the schedule generated in the first step.

This process repeats whenever a new task arrives, which incurs a time complexity of $O(M^2N)$, where M is the number of tasks in the ready queue and N is the number of frequency levels supported by the processor.

The above works adopt a design strategy of decoupling the energy constraint and timing constraint to meet system requirements and improve system performance. However, the energy constraint and time constraints interact, and decoupling the two design constraints may unnecessarily lead to task deadline violation. Scheduling a task at the lowest possible operating frequency stretches the execution of the task, which essentially steals the time quota of the remaining tasks. Some of the remaining tasks may miss their deadlines due to lack of time quota, even if there are plenty of energy.

Imagine a system that contains a processor supporting two frequency levels ($f_1 = 500$ Hz and $f_2 = 1000$ Hz) and has three real-time tasks. The arrival time, absolute deadline, and execution cycles of each task are given by a triple, that is, τ_1: {0, 1.2, 500}, τ_2: {0, 1.5, 250}, and τ_3: {1, 1.5, 300}. The task schedule generated by the existing schemes [17] is shown in Fig. 1(a). At $t = 0$, task τ_1 and τ_2 are in the ready queue and are scheduled to run at $f = 500$ Hz. τ_1 is supposed to finish at $t = 1$, τ_2 is supposed to finish at $t = 1.5$, and the two tasks meet their deadlines. At $t = 1$, the execution of task τ_1 is finished, τ_3 arrives, and the task schedule is updated. Task τ_2 and τ_3 are in the ready queue and are scheduled to run at $f = 1000$ Hz. Task τ_2 is supposed to finish its execution at $t = 1.25$ and task τ_3 misses its deadline at $t = 1.5$. The timeline at the bottom of Fig. 1(a) illustrates the execution of the three tasks.

The proposed scheme considers the scheduling of tasks in a period of time called feasible interval. As is illustrated in Fig. 1(b), the feasible interval at $t = 0$ contains all the three

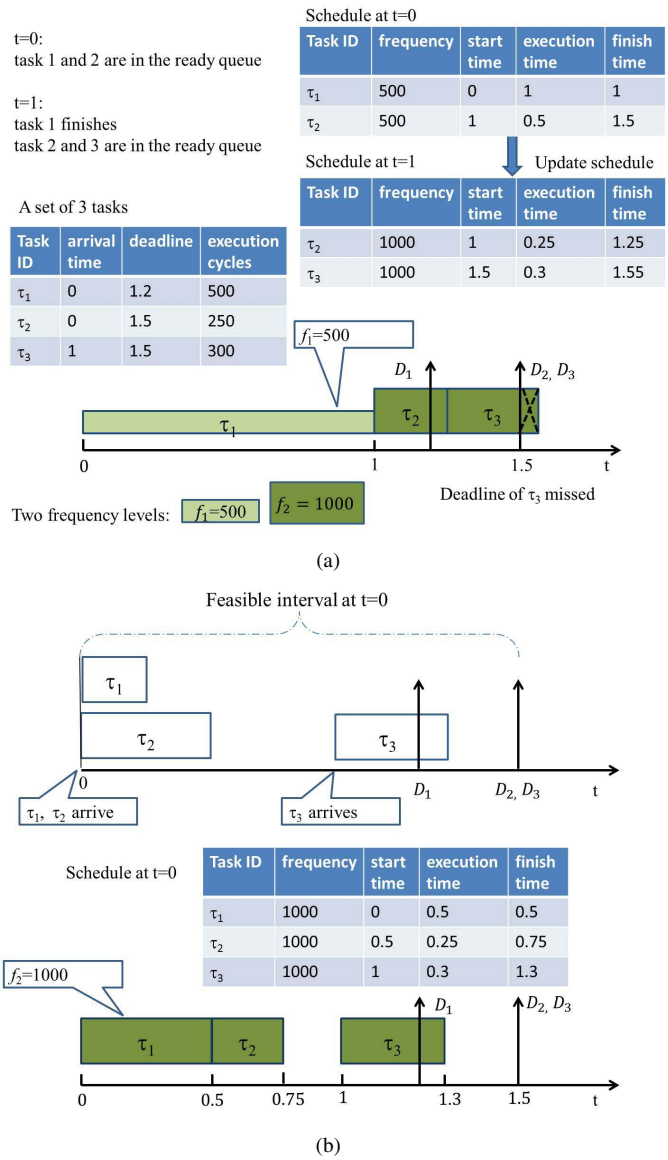


Fig. 1. Motivation example. (a) Scheduling strategy of existing schemes [17]. (b) Scheduling strategy of the proposed schemes.

tasks, which are supposed to operate at $f = 1000$ Hz. The task schedule at $t = 0$ indicates that the task τ_1, τ_2, and τ_3 are supposed to finish their execution at $t = 0.5$, 0, 75, and 1.3, respectively. The timeline at the bottom of Fig. 1(b) indicates that all the three tasks meet their deadlines.

In this paper, the authors propose a dynamic scheduling scheme that jointly considers the system energy availability and system utilization state. The proposed scheme first characterizes the state of the target system from the perspectives of system utilization and harvested energy. It then allocates a portion of the available energy to the group of tasks in the ready queue based on the amount of energy produced by the PV array and the value of system utilization. The operating frequency for the group of task is selected according to the allocated energy and the selected frequency is fine tuned to further improve energy efficiency. It is also noteworthy that the proposed scheme is much simpler when compared with the existing schemes. Its time complexity is

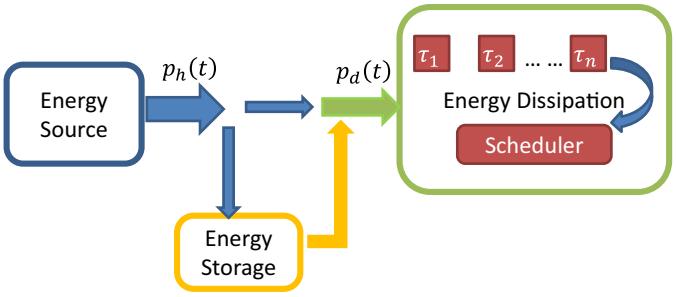


Fig. 2. Architecture diagram of an energy-harvesting real-time system.

one order of magnitude lower than that of the benchmarking scheme. Experimental results show that the proposed dynamic scheduling scheme achieves a comparable deadline miss rate and higher energy efficiency as compared with the state-of-the-art schemes.

The rest of this paper is organized as follows. Section II introduces system architecture and models. Section III characterizes the system in terms of system utilization state and system energy state, and describes the proposed global energy allocation strategy based on the two states. The state-aware frequency selection algorithm is proposed in Section IV. Section V presents the experimental results and analysis, and Section VI concludes this paper.

II. System Architecture and Models

The target energy-harvesting real-time system consists of three major modules: the energy source module, energy storage module, and energy dissipation module. As is described in Fig. 2, the energy source module scavenges energy from the environment at the power of $p_h(t)$, and the energy dissipation module drains energy from either the energy source or storage module at the rate of $p_d(t)$. When the harvested energy is more than the energy consumed by the dissipation module, the extra energy is stored in the energy storage module. The typical example of an energy dissipation module is a processor that schedules and executes real-time applications.

A. Energy Model

The energy source module converts the renewable generation such as solar, thermal, and wind energy to electrical energy. Let $p_h(t)$ denote the harvesting power and $E_h(t_1, t_2)$ denote the energy scavenged from the environment during the interval $[t_1, t_2]$, then $E_h(t_1, t_2)$ is expressed as

$$E_h(t_1, t_2) = \int_{t_1}^{t_2} p_h(t)dt. \tag{1}$$

Part of the harvested energy is consumed by the microprocessor and the residuals are stored in the energy storage module. For the case of solar energy, the energy source module is a set of solar panels that outputs the energy of $E_h(t_1, t_2)$ during the interval $[t_1, t_2]$. The energy storage module is typically in the form of a super capacitor or a battery. The battery capacity is in general deemed to be limited. Let E_{cap} denote the capacity of a battery, and $E(t)$ denote the residual energy in the battery at the time instance t, then the inequality $E(t) \leq E_{\text{cap}}$ holds.

The target processor is an example of energy dissipation module. The power consumption of a CMOS device can be modeled as the sum of dynamic power consumption and static power consumption. The average dynamic power consumption p_d can be estimated by a strictly increasing and convex function, that is, $p_d \propto f^3$ [18]. Let p_s denote the static power consumption of a device, I_{subn} denote the sub-threshold leakage current, and I_j denote the reverse bias junction current, then the static power consumption of the device is given by $p_s = V_{\text{dd}} I_{\text{subn}} + |V_{\text{bs}}| I_j$, where V_{bs} is the body bias voltage, and V_{dd} is the supply voltage [19]. The total energy (E_{demand}) consumed by a processor during an interval $[t_1, t_2]$ when a real-time application executes is hence estimated by

$$E_{\text{demand}} = (t_2 - t_1) \times p_s + \int_{t_1}^{t_2} p_d(t)dt. \tag{2}$$

B. Power Prediction Model

It has been shown that accurate prediction of the harvesting power is crucial to the efficiency of the optimization techniques for energy-harvesting systems. Power prediction models have been extensively investigated in [20]–[23]. In [20], the authors investigated autoregressive model, neural networks model, and the adaptive-network-based fuzzy inference system (ANFIS) model. The three models were compared with predict half-daily values of global solar irradiance with a temporal horizon of three days. Khatiba *et al.* [21] reviewed the linear, nonlinear, and artificial intelligence modeling techniques, and compared the three models in terms of the prediction accuracy. In particular, the artificial neural network models were utilized to model and predict the solar energy for different areas in the world in [22] and [23].

In this paper, the exponential moving average approach [24] is adopted to predict the harvesting power of the system. The exponential moving average is a forecasting technique that applies the exponentially decreasing factors to time series datum. The exponential moving average for a series can be calculated recursively, that is

$$\hat{x}_t = \alpha \times x_t + (1 - \alpha) \times \hat{x}_{(t-1)} \tag{3}$$

where $\hat{x}_t$ is the exponentially smoothed estimate at the time instance t, x_t is the observed value at the same time, $\alpha (0 \leq \alpha \leq 1)$ is the degree of weighting decrease, and $\hat{x}_{(t-1)}$ is the smoothed estimate at the previous time instance. The exponential moving average approach is adopted in the proposed scheme to estimate both the long-term and the short-term harvesting powers. Since it assigns a larger weight to the more recent data, the short-term estimate is more accurate when compared with the long-term estimate.

C. Task Model

Consider a scenario that real-time tasks in a ready queue are ordered using the EDF policy, and the task at the head of the ready queue is to be executed on a DVS-capable processor. It is assumed that the DVS-capable processor supports N discrete operating frequencies $\{f_1, \ldots, f_N\}$, and $f_i < f_j$ holds for $i < j$. The power consumption of the processor varies with

frequency scaling. Let p_i indicate the power consumption of the processor at the frequency f_i, then the power consumption of the processor at different frequency levels can be modeled as $\{p_1, \ldots, p_N\}$.

The timing characteristics of a task τ_i is modeled using a triplet $\tau_i : \{a_i, d_i, c_i\}$, where a_i is the arrival time, d_i is the relative deadline, and c_i is the worst case execution cycles of the task. The absolute deadline of task τ_m can be expressed as $a_m + d_m$. Let st_i is the start time to execute the task, and assume that the task is scheduled to be executed at the frequency f_k, then the execution of the task cannot start until it arrives, and must finish before its absolute deadline, as is described by the inequality

$$\begin{cases} a_i \leq st_i \\ st_i + \frac{c_i}{f_k} \leq a_i + d_i \end{cases} \tag{4}$$

where c_i/f_k gives the worst case execution time of the task. Note that the task τ_i can be either a periodic task or an aperiodic task, and the proposed scheme can handle the scheduling of both types of tasks. Furthermore, it is assumed that the execution of a real-time task is preemptable. That is, when a task of higher priority arrives, the execution of the current task of lower priority is preempted. When the preempted task is resumed for execution, it may execute at the same or different processor speed, depending upon the system state.

III. Proposed Global Energy Allocation Strategy

The energy and timing design constraints are decoupled in [17] to reduce design complexity. In [17], when a new task is coming it is first push into a queue. The real-time tasks in the queue are then scheduled at the lowest possible processor speed and the workload is evenly (with respect to time) allocated to the processor. The energy constraint is considered when the task schedule satisfying the timing constraints is generated. All the available energy is allocated to the task being scheduled, and if the available energy is not enough to finish the task, it is dropped and removed from the queue. This process repeats whenever a new task comes, which results in a high design complexity that is not well-suited for dynamic scheduling.

On one hand, on assigning a large portion or if even all of the available energy to the task being scheduled increases, the possibility that remaining tasks miss their deadlines. This is because the task consumes too much energy and remaining tasks may not have enough energy to finish their execution. On the other hand, assigning a small portion of available energy to the task being scheduled will prolong its execution since it has to execute at a relatively low frequency. The current task consumes extra CPU time and the chances that the remaining tasks miss their deadlines increase even if there are plenty of energy. In fact, energy and timing design constraints interact, hence need to be jointly optimized instead of being decoupled. This section describes the proposed global energy allocation strategy that enhances energy efficiency and reduces task deadline miss rate by considering the overall system workload and energy availability. The proposed energy allocation policy assigns available energy to a period of time based on the system utilization state and the system energy state, as is detailed below.

A. System Utilization State

The system utilization is defined to be the sum of utilizations of all tasks in the system. The state of a system can be described by its utilization. When the utilization of a system exceeds a threshold value, real-time tasks in the system have higher probability of missing their deadlines due to a small-scale variation in available energy.

The system is deemed to be overloaded in this case. When a system is in the overload state, a portion of energy in the battery is allocated for the system to mitigate the overload of the workload. This strategy tries to improve the efficiency of energy and reduce deadline miss rate of real-time tasks. It is clear that two issues need to be addressed in system state characterization. The first one is to derive the threshold value that defines the overload state of the system, and the second one is to calculate the system utilization with respect to a certain interval.

Instead of being fixed, the threshold value essentially changes dynamically within a range to better adapt to the system state. Let U_{th} denote the threshold value, and $U_{\text{th,L}}$ and $U_{\text{th,H}}$ denote the upper bound and lower bound of the threshold, respectively, then U_{th} is in the range of $[U_{\text{th,L}}, U_{\text{th,H}}]$. The upper bound $U_{\text{th,H}}$ is typically set to 1. The lower bound $U_{\text{th,L}}$ is determined by characteristics of real-time tasks in the task set. For periodic tasks, an offline task schedule generated by any energy efficient task-scheduling algorithms gives the optimum or suboptimum operating frequencies of tasks. Further stretching tasks and increasing system utilization by scaling down the operating frequencies results in with high probability an increase in task deadline miss rate. Therefore, the system utilization of an offline task schedule, which is calculated as the sum of utilizations of all tasks in the task set, is defined to be the lower bound on the threshold value. Since the arrival time of an aperiodic task is not predictable, the lower bound on the threshold value $U_{\text{th,L}}$ is set to 0 for aperiodic tasks. When the task set contains both periodic and aperiodic tasks, the $U_{\text{th,L}}$ is determined by characteristics of periodic tasks.

Given the upper bound and lower bound, the utilization threshold value U_{th} can be easily derived. The U_{th} is initialized by randomly picking a value in $[U_{\text{th,L}}, U_{\text{th,H}}]$. Algorithm 1 describes the procedure to dynamically update the threshold value U_{th} that demarcates the boundaries of dynamic system utilization states. On one hand, if a task is dequeued due to lack of time, that is, if the system utilization is high and the task cannot finish the execution before its deadline, the threshold value U_{th} is decreased by one step. The decrement of U_{th} indicates that the system is more likely to switch to the overloaded state, thus, more energy is to be allocated to mitigate the overloaded state, which in turn improves the timing of real-time tasks. On the other hand, if a task is dequeued due to lack of energy, that is, if the available energy is not enough for the task to finish the execution before its

Algorithm 1 Dynamically Update the Threshold Value U_{th} That Demarcates the Boundaries of Dynamic System Utilization States

Require: Upper bound $U_{th,H}$ and lower bound $U_{th,L}$ on the threshold value U_{th}

1: **if** a task is dequeued due to lack of time **then**
2: $U_{th} = U_{th}$ - 0.01;
3: **end if**
4: **if** a task is dequeued due to lack of energy **then**
5: $U_{th} = U_{th}$ + 0.01;
6: **end if**
7: **if** $U_{th} \geq U_{th,H}$ **then**
8: $U_{th} = U_{th,H}$;
9: **end if**
10: **if** $U_{th} \leq U_{th,L}$ **then**
11: $U_{th} = U_{th,L}$;
12: **end if**
13: **return**

absolute deadline, the threshold value U_{th} is increased by one step. The increment of U_{th} indicates that the system is less likely to switch to the overloaded state, hence less battery energy is to be allocated to future tasks, which in fact scales down the processor speed and conserves energy for future tasks. The step size for threshold adaptation is an empirical value, and 0.01 is taken in Algorithm 1. In the runtime, when the system utilization with respect to a certain interval is greater than or equal to the U_{th}, the system is deemed to be in the overload state in the period. Otherwise, it is not overloaded in this duration.

The system utilization at a time instance is defined with respect to a certain interval. Suppose a queue named Q_1 contains tasks that are ready to execute at time instance t. Let $D_{\max}$ denote the maximum of the latest finish times of all tasks in the queue Q_1, then the interval $[t, D_{\max}]$ is defined to be the feasible interval at the time instance t. It is clear that all tasks in the queue Q_1 finish their execution in the feasible interval. It is likely that new tasks arrive in the feasible interval of $[t, D_{\max}]$. Those tasks that arrive and finish their execution in the feasible interval $(t, D_{\max}]$ are push into a queue named Q_2, and those tasks that arrive in the interval of $(t, D_{\max})$ but finish their execution beyond the time instance $D_{\max}$ are pushed into a queue named Q_3. Note that Q_1 is a real queue that contains ready tasks while Q_2 and Q_3 are virtual queues that are utilized to illustrate the concept of the feasible interval. Fig. 3 illustrates the feasible interval at the time instance t. Three tasks of τ_1, τ_2, and τ_3 of absolute deadlines of D_1, D_2, and D_3 are ready to be executed at the instance t. The maximum of the latest finish time of the three tasks is $D_{\max} = D_2$. Thus, the feasible interval at the instance t is $[t, D_{\max}]$. It is also shown in Fig. 3 that the task τ_4, τ_5, and τ_6 arrive and finish their execution in the feasible interval $(t, D_{\max}]$ while the task τ_7 and τ_8 arrive in the feasible interval but do not have to finish their execution in the feasible interval. The task τ_1, τ_2, and τ_3 are push into the queue Q_1, the task τ_4, τ_5, and τ_6 are push into the queue Q_2, and the task τ_7 and τ_8 are push into the queue Q_3.

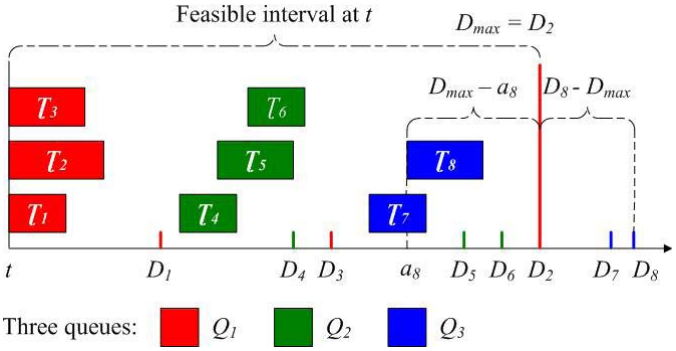


Fig. 3. Illustration of feasible interval at the instance t.

The system utilization with respect to the feasible interval $[t, D_{\max}]$ is calculated by considering the tasks in the queues Q_1, Q_2, and Q_3. The tasks in Q_1 and Q_2 finish their execution in the feasible interval $[t, D_{\max}]$, thus, their contribution to the system utilization at the instance t is determined.

Two cases are considered for tasks in the queue Q_3. In the first case, the $D_{\max}$ is deemed to be an equal ratio point of the execution time of task $\tau_i \in Q_3$ with respect to the feasible interval (a_i, D_i) of the task, where a_i and D_i are the arrival time and absolute deadline of the task τ_i, respectively. In other words, the ratio of the portion of the task τ_i being executed before $D_{\max}$ to the portion of the task being executed after $D_{\max}$ is assumed to equal the ratio of $(D_{\max} - a_i)$ to $(D_i - D_{\max})$. As is shown in Fig. 3, the feasible interval $[a_8, D_8]$ of the task τ_8 is divided by $D_{\max}$ into two periods, $[a_8, D_{\max}]$ and $[D_{\max}, D_8]$, and the portions of the task executed in the two periods are proportionate with respect to the length of the two periods. This scenario can be viewed as an average case of task execution. In the second case, tasks in the queue Q_3 is assumed to start execution after the time instance $D_{\max}$, and only tasks from queues Q_1 and Q_2 will be executed in the interval $[t, D_{\max}]$. As is shown in Fig. 3, the task τ_8 will be executed in the interval of $[D_{\max}, D_8]$ in this case and the workload in the interval of $[t, D_{\max}]$ is thus reduced to minimum.

Assume that tasks in the interval $[t, D_{\max}]$ operate at the frequency f_k, and let $U(t, D_{\max}, f_k)$ denote the system utilization with respect to the feasible interval $[t, D_{\max}]$ at the frequency f_k, then $U(t, D_{\max}, f_k)$ is given by

$$U(t, D_{\max}, f_k) = \frac{\sum_{\forall \tau_i \in Q_1} \frac{c_i}{f_k} + \sum_{\forall \tau_i \in Q_2} \frac{c_i}{f_k} + \sum_{\forall \tau_i \in Q_3} \frac{c_i}{f_k}}{D_{\max} - t}. \quad (5)$$

With respect to the feasible interval $[t, D_{\max}]$, (5) gives the system utilization. All tasks in queues Q_1, Q_2, and Q_3 are predetermined for periodic systems, however, tasks in queues Q_2 and Q_3 are not known at the time instance t for aperiodic systems. A simple yet efficient method is designed in this paper to estimate tasks in queues Q_2 and Q_3 for aperiodic systems. The method employs historical data to predict the total utilizations of tasks in queues Q_2 and Q_3. More specifically, based on characteristics of tasks over a past period of time, the method first picks a time instance

when task scheduling is performed, determines the feasible interval at the time instance, and finds the aperiodic tasks in queues Q_2 and Q_3 of the interval. It then calculates the total utilization of tasks in queue Q_2 as $\sum_{\forall \tau_i \in Q_2} c_i/f_k$ and the total utilization of tasks in queue Q_3 as $\sum_{\forall \tau_i \in Q_3} c_i/f_k$, where f_k is the processor operating frequency. Finally, the method collects a set of consecutive time instances when task scheduling is performed, applies the exponential moving average technique to the set to obtain average utilizations for queues Q_2 and Q_3, and takes these values as the estimation for utilizations of queues Q_2 and Q_3 in the current feasible interval $[t, D_{\max}]$.

The utilization $U(t, D_{\max}, f_k)$ combined with the threshold value U_{th} for system utilization state demarcation is utilized to define the dynamic system utilization state that determines the allocation policy of the available energy, as is detailed in Section III-C.

B. System Energy State

The energy state of a system can be characterized by the average amount of the harvested energy with respect to different periods of time. More specifically, when the average harvested energy with respect to a short period is less than the average harvested energy with respect to a long period, the system is deemed to be in low-energy state. Let $p_{s\_\text{solar}}(t)$ and $p_{l\_\text{solar}}(t)$ denote the harvesting power of the system with respect to a short and long period of time, respectively, and let $E_{s\_\text{solar}}(t, D_{\max})$ and $E_{l\_\text{solar}}(t, D_{\max})$ denote the harvested energy in the interval of $[t, D_{\max}]$ based on the harvesting power of the short and long period, respectively. The harvested energy $E_{s\_\text{solar}}(t, D_{\max})$ and $E_{l\_\text{solar}}(t, D_{\max})$ are then expressed as the integral of the harvesting power during the feasible interval of $[t, D_{\max}]$, as is given by (6) and (7)

$$E_{s\_\text{solar}}(t, D_{\max}) = \int_t^{D_{\max}} p_{s\_\text{solar}}(t)dt \tag{6}$$

$$E_{l\_\text{solar}}(t, D_{\max}) = \int_t^{D_{\max}} p_{l\_\text{solar}}(t)dt. \tag{7}$$

It is clear that when $E_{s\_\text{solar}}(t, D_{\max}) \leq E_{l\_\text{solar}}(t, D_{\max})$, the system is in the low-energy state.

Note that at the time instance t the harvesting power of the system during the feasible interval $[t, D_{\max}]$ is not known in advance. The power prediction models described in Section II-B are thus utilized to derive the short-term power $p_{s\_\text{solar}}(t)$ and the long-term power $p_{l\_\text{solar}}(t)$.

C. Proposed Energy Allocation Strategy

The system energy available at the time instance t consists of the energy produced by the PV panel and the energy stored in the battery. The battery is essentially an energy buffer for the system. On one hand, the energy produced by the PV panel is directly consumed by the system, and the excess energy is stored in the battery. On the other hand, the system derives energy from the battery when it is in the low-energy state or in the overload state. This subsection focuses on the allocation policy for the battery energy based on the system utilization state and system energy state.

As is described in Section III-A, a system is deemed to be in the overloaded state when the system utilization with respect to the interval $[t, D_{\max}]$ is greater than or equal to the system utilization threshold, that is, $U(t, D_{\max}, f_k) \geq U_{\text{th}}$ holds, where f_k is the operating frequency of the system and U_{th} is the system utilization threshold. A fraction of the battery energy is allocated to the feasible interval $[t, D_{\max}]$ when the system is in the overloaded state. The amount of the battery energy allocated to the feasible interval should be large enough for the system to sustain its operation in the interval at the lowest processor speed f_1 when it is extremely overloaded. The battery energy allocated to the feasible interval $[t, D_{\max}]$, which is denoted by E_1, is thus given by

$$E_1 = \begin{cases} \Delta U \times (D_{\max} - t) \times p_1, & \Delta U \geq 0 \\ 0, & \Delta U < 0 \end{cases} \tag{8}$$

where the term $\Delta U = U(t, D_{\max}, f_k) - U_{\text{th}}$ is the degree of the system being overloaded, the term $(D_{\max} - t)$ 1 s the length of the interval, and p_1 is the processor power consumption at the lowest operating frequency f_1. When the system is strikingly overloaded, that is, ΔU is becoming large, the energy allocated to the interval $[t, D_{\max}]$ will also increase. It can be seen from the equation that the allocated battery energy E_1 adapts to the degree to which the system is overloaded.

Similarly, a fraction of the battery energy is allocated to the feasible interval $[t, D_{\max}]$ when the system is in the low-energy state. The system energy state is defined and described in Section III-B. The amount of the battery energy allocated to the feasible interval should also be large enough for the system to sustain its operation in the interval at the lowest processor speed f_1 when it is in very low-energy state. The battery energy allocated to the feasible interval, which is denoted by E_2, is thus given by

$$E_2 = \begin{cases} \Delta E_{\text{solar}} \times (t - D_{\max}) \times p_1, & \Delta E_{\text{solar}} \leq 0 \\ 0, & \Delta E_{\text{solar}} > 0 \end{cases} \tag{9}$$

where the term $\Delta E_{\text{solar}} = (E_{s\_\text{solar}} - E_{l\_\text{solar}})/E_{l\_\text{solar}}$ is the degree to which the system is in low-energy state, the term $(D_{\max} - t)$ is the length of the interval, and p_1 is the processor power consumption at the lowest operating frequency f_1. The E_{s_solar} and E_{l_solar}, which are given in (6) and (7), respectively, are the predicted short-term solar energy and predicted long-term solar energy in the feasible interval $[t, D_{\max}]$. The battery energy E_2 is allocated to the feasible interval when $\Delta E_{\text{solar}} \leq 0$ holds, that is, when the system is in low-energy state; no battery energy is allocated to the interval otherwise. The predicted solar energy E_{s_solar} and E_{l_solar} are functions of the average harvesting power p_{s_solar} and p_{l_solar}, respectively, thus the allocated battery energy E_2 is closely related to the average harvesting power.

Considering the system utilization state and the system energy state, the battery energy allocated to the feasible interval $[t, D_{\max}]$, which is denoted by E_{alloc}, is thus expressed as

$$E_{\text{alloc}} = E_1 + E_2 \tag{10}$$

where E_1 and E_2 are given by (8) and (9), respectively. It is clear that the allocated battery energy (E_{alloc}) adapts to the system utilization state and system energy state.

Note that the battery energy is allocated to a feasible interval where more than one task are likely to execute. This strategy regards tasks in the feasible interval as a group when allocating battery energy and performs the energy allocation computation whenever a new task arrives. It allocates a portion of the available battery energy to a feasible interval where multiple tasks are likely to execute by considering both the system utilization and energy state, which effectively enhances the energy efficiency and system performance, as is described in the next section.

IV. State-Aware Dynamic Frequency Selection Algorithm

The proposed state-aware dynamic frequency selection algorithm aims at achieving good energy efficiency at a low computational complexity. The algorithm selects an operating frequency for tasks in the feasible interval $[t, D_{\max}]$ based on the energy allocated to the interval. It first finds the lowest possible operating frequency for the current task, then selects the operating frequency for the feasible interval based on the system utilization state and energy state, and finally adjusts the operating frequency when the solar energy produced by the PV panel is excessive. The following subsections describe the three steps in details.

A. Frequency Selection Based on Utilization and Energy State

As is described in Section II-C, the target DVS-capable processor is assumed to support N discrete operating frequencies. Let τ_m be the task at the head of the ready queue, and let f_{low} be the lowest operating frequency for the task without violating the timing constraint. The task can finish the execution before its absolute deadline if the following holds:

$$a_m + d_m - \frac{c_m}{f_{\text{low}}} \geq \max\{a_m, t\} \tag{11}$$

where a_m, d_m, and c_m are the arrival time, relative deadline, and execution cycles of the task, respectively, and t is the current time instance. Since the a_m, d_m, c_m, and t are all known, the lowest processor frequency f_{low} that meets the timing constraint of the task can be derived using the equation.

Considering the maximum processor-supported frequency $f_{\max} = f_N$, the task can execute at any frequency between f_{low} and $f_{\max}$. On one hand, when the task executes at a low operating frequency, its execution is prolonged, which may lead to deadline violation of remaining tasks due to lack of time to finish the execution. On the other hand, when the task executes at a high operating frequency, it consumes a large portion of energy, which may lead to deadline violation of remaining tasks due to lack of energy. As a result, a frequency selection algorithm that jointly considers the system energy state and utilization state in a feasible interval is needed to identify the proper operating frequency for the task.

The state-aware frequency selection procedure given in Algorithm 2 is expected to produce a processor-supported frequency that jointly considers energy and time to improve system performance and enhance energy efficiency. The desired

Algorithm 2 Select the Operating Frequency Based on System Utilization and Energy State

Require: Task τ_m at the head of the ready queue Q
1: $D_{max} = \max\{(a_k + d_k) : \tau_k \in Q\}$;
{select frequency that guarantees the timing constraint of τ_m}
2: derive the lowest frequency f_{low} using Equation (11);
3: **for** $i = N$ to 1 **do**
4: $select = i$;
5: **if** $f_i < f_{low}$ **then**
6: $select = N + 1$;
7: **break**;
8: **end if**
9: calculate energy supply E_{supply} for feasible interval $[t, D_{max}]$ using Equation (12);
10: calculate energy demand E_{demand} for feasible interval $[t, D_{max}]$ using Equation (13);
11: **if** $E_{supply} \geq E_{demand}$ **then**
12: **break**;
13: **end if**
14: **end for**
{check time and energy availability for task τ_m}
15: **if** $select > N \parallel \left(\frac{(p_d + p_s) \times c_m}{f_{select}} > E(t) + E_{s\_solar}(t, a_m + d_m)\right)$ **then**
16: remove task τ_m from the ready queue Q;
17: set task τ_m to be null;
18: **return**;
19: **end if**
20: **return** f_{select};

frequency is denoted by f_{select}, where select is defined to be the index of the frequency. The algorithm takes as input the task τ_m at the head of the ready queue Q. It first derives the maximum absolute deadlines of all tasks in the ready queue ($D_{\max}$), which gives the upper bound of the feasible interval $[t, D_{\max}]$ (line 1). It then calculates the lowest processor frequency f_{low} that guarantees the timing constraint of the task τ_m using (11) (line 2). The system could be in overload state or low-energy state in the feasible interval. The battery energy is allocated to the feasible interval when the system is in either state, as is given by (10). In addition to the battery energy, the system consumes solar energy generated by the PV panel, which is given by (6). Note that instead of the long-term predicted solar energy E_{l_solar}, the short-term predicted solar energy E_{s_solar} given in (6) is utilized to estimate the solar energy fed to the system. This is because the short-term prediction is more precise when compared with the long-term estimation. The energy supplied to the feasible interval, which is denoted by E_{supply}, is hence expressed as the sum of the allocated batter energy and the estimated solar energy, as is given by

$$E_{\text{supply}} = E_{\text{alloc}} + E_{s\_\text{solar}} \tag{12}$$

where E_{alloc} and E_{s_solar} is given in (10) and (6), respectively. Since E_{alloc} is closely related to the harvesting power and

E_{s_solar} is an integral function of the harvesting power, the supply energy E_{supply} adapts to the harvesting rate of the system.

The energy demand for the feasible interval $[t, D_{max}]$, which is denoted by E_{demand}, is defined as the sum of the static energy and the dynamic energy consumed during the interval. Equation (2) gives the energy demand for the interval, and is rewritten as

$$E_{demand} = (D_{max} - t) \times p_s + \int_t^{D_{max}} p_d(t)dt \qquad (13)$$

where p_s is the static power consumption and $p_d(t)$ is the dynamic power consumption of the processor.

The algorithm proceeds to identify the operating frequency for the current task from lines 3 to 14. It starts from the maximum operating frequency $f_{max} = f_N$, examines each frequency, and breaks the loop when the lowest frequency f_{low} is reached (lines 4–8) or when the desired frequency that jointly considers the energy and time is selected (lines 9–13). Since the lowest frequency obtained using (11) may not be the processor-supported frequency, the lines 5–8 of the algorithm map this frequency to the processor-supported frequency. The algorithm calculates the energy supply in line 9, calculates the energy demand in line 10, and compares the energy supply and energy demand in line 11. If $E_{supply} \geq E_{demand}$ holds, the current frequency is taken as the operating frequency for the task. Otherwise, the algorithm proceeds to the next frequency level to repeat the process.

If the processor frequency f_{select} is selected, the timing constraint of the task is satisfied but the system may not have enough energy to finish the execution of the task. The energy needed for the task τ_m: $\{a_m, d_m, c_m\}$ to finish the execution at f_{select} is expressed as

$$\frac{(p_d + p_s) \times c_m}{f_{select}}$$

and the energy available for the task τ_m is given by

$$E(t) + E_{s\_solar}(t, a_m + d_m)$$

where p_d and p_s are the processor dynamic power consumption and static power consumption, respectively. $E(t)$ and $E_{s\_solar}(t, a_m + d_m)$ are the battery energy at the instance t and the harvested solar energy in the duration of $[t, a_m + d_m]$, respectively. If the amount of the needed energy is greater than that of the available energy, the task cannot finish the execution before its absolute deadline. It is then removed from the queue, and the task variable τ_m is set to null, as is shown from lines 15 to 19.

The schedulability of the current task being scheduled (τ_m) is guaranteed since f_{select} is limited in the range of $[f_{low}, f_{max}]$. In fact, the algorithm aims at selecting a frequency for the task at the head of the ready queue. It considers the impact of scheduling the current task on remaining tasks from aspects of both energy and time. All tasks in the ready queue are taken into account when the energy supply and demand with respect to the feasible interval are calculated, and it is assumed that these tasks will execute at the same frequency f_{select}. This strategy reduces the probability that the current task consumes too much energy and time so that the remaining tasks will miss their deadlines due to lack of energy or time. The remaining tasks in the ready queue may not be schedulable at the f_{select}, their schedulability will be verified by calling the frequency selection algorithm when they arrive at the head of the ready queue.

Algorithm 3 Fine-Tune the Selected Frequency f_{select} When Overflowing and Derives the Optimal Start Time for Task τ_m

Require: The selected frequency f_{select} for task τ_m
1: **if** task τ_m is null **then**
2: **return** ;
3: **end if**
{fine-tune the frequency f_{select} when overflow occurs}
4: **if** $E(t) + E_{s\_solar}(t, D_{max}) - E_{demand} > E_{cap}$ **then**
5: **for** $i = select$ to N **do**
6: update energy demand E_{demand} via Equation (13);
7: **if** $E(t) + E_{s\_solar}(t, D_{max}) - E_{demand} \leq E_{cap}$ **then**
8: $select = i$;
9: **break**;
10: **end if**
11: **end for**
12: **end if**
13: calculate the start time st_m for task τ_m using Equation (14);
14: **return** st_m;

B. Fine-Tune the Selected Frequency and Derive the Start Time for Current Task

One of the major goals of designing an energy-harvesting system is to efficiently utilize the renewable energy. As far as the proposed design is concerned, the solar energy produced by the PV panel is directly fed to the microprocessor, and the excess energy is stored in a battery. The battery capacity is in general limited, and a battery may not be able to store all the excess solar energy. This necessitates the fine-tuning of the selected operating frequency to enhance energy efficiency.

The energy of the system overflows when the difference between the available energy and the energy demand exceeds the battery capacity. In other words, when the excess energy is too much such that it cannot be stored in the battery, the overflow occurs. One effective approach to eliminate energy overflow is to increase the operating frequency of the system, which can both utilize excess solar energy and generate slack time for remaining tasks.

Algorithm 3 derives the fine-tuned operating frequency f_{select} (lines 4–12) and calculates the optimal start time for the current task using 14 (line 13). It first checks whether the current task τ_m is null (lines 1–3). It then checks whether the system energy overflows. Let E_{cap} denote the energy capacity of the battery in the target system. If $E(t) + E_{s\_solar}(t, D_{max}) - E_{demand} > E_{cap}$ holds, the overflow occurs. In this case, the algorithm increases the operating frequency by one level, updates the energy demand E_{demand} using (13), and checks

whether the energy still overflows (lines 5–11). This process is repeated until the overflow is eliminated or the frequency $f_{\max} = f_N$ is reached without eliminating the overflow. In the later case, the $f_{\max}$ is taken as the operating frequency for the current task.

After fine-tuning the operating frequency, the algorithm derives the optimal start time for the current task, which is achieved using the LSA [12]. The LSA proposed by Moser *et al.* [12] is designed to schedule real-time tasks for energy-harvesting system under the EDF scheduling policy. It postpones the execution of a task as late as possible under the assumption of a constant operating frequency.

Let st_m denote the optimal start time of task τ_m, then st_m can be derived using the LSA scheduling algorithm, and is given by

$$st_m = \max\{st_m^*, st_m^{'}\} \tag{14}$$

where st_m^* is given by

$$st_m^* = a_m + d_m - \frac{E(t) + E_{s\_\text{solar}}(t, a_m + d_m)}{p_d + p_s}$$

and $st_m^{'}$ can be derived using the equation

$$E_{s\_\text{solar}}(t, st_m^{'}) - E_{\text{cap}} = E_{s\_\text{solar}}(t, a_m + d_m) + (st_m^{'} - a_m - d_m)(p_d + p_s).$$

The a_m, d_m, p_d, and p_s are the arrival time, relative deadline, dynamic power consumption, and static power consumption of the task, respectively. $E(t)$ is the energy in storage module at the time instance t, $E_{s\_\text{solar}}(t, a_m + d_m)$ is the harvested energy from the instance t to the absolute deadline $(a_m + d_m)$ of the task τ_m and $E_{s\_\text{solar}}(t, st_m^{'})$ is the harvested energy from the instance t to the instance $st_m^{'}$. E_{cap} is the battery capacity.

C. State-Aware Frequency Selection Algorithm

Algorithm 4 outlines the proposed state-aware frequency selection algorithm by assembling the above described major steps given in Algorithms 1–3. The system utilization threshold U_{th} is initialized by randomly picking a value in $[U_{\text{th,L}}, U_{\text{th,H}}]$, where $U_{\text{th,L}}$ and $U_{\text{th,H}}$ are, respectively, the lower bound and upper bound on the system utilization threshold, which is described in Section III-A. The algorithm maintains a ready queue Q that is sorted based on the EDF scheduling policy, that is, the task of the earliest deadline is kept at the head of the queue and is assigned the highest priority. If the ready queue Q is not empty, the algorithm first picks the task at the head of the queue as the current task τ_m and updates the utilization threshold U_{th} using Algorithm 1. It then selects a proper operating frequency by jointly considering the system utilization and energy state in the feasible interval (Algorithm 2). The selected operating frequency is fine tuned when overflow in energy occurs, and the optimal start time of the task is derived using the LSA (Algorithm 3). If the task τ_m is not null, it will execute at the scheduled start time and frequency. The task is removed from the ready queue and the τ_m is reset to null when the execution is finished.

Algorithm 4 State-Aware Frequency Selection Algorithm

Require: Maintain a ready task queue Q
1: set the queue Q empty;
2: Initialize the system utilization threshold U_{th};
3: **while** true **do**
4: **if** new task arrives **then**
5: push the task into Q;
6: sort Q in the increasing order of task deadlines;
7: **end if**
8: **if** Q is not empty **then**
9: pick the task at the head of Q as current task τ_m;
10: update utilization threshold U_{th} using Algorithm 1;
11: select the state-aware frequency using Algorithm 2;
12: fine-tune the selected frequency using Algorithm 3;
13: calculate start time of the task using Algorithm 3;
14: **end if**
15: **if** task τ_m is not null **then**
16: execute the current task τ_m;
17: **end if**
18: **if** the execution is finished **then**
19: remove the task from the queue Q;
20: set task τ_m to be null;
21: **end if**
22: **end while**

The time complexity of Algorithms. 1–3 are $O(1)$, $O(MN)$, and $O(N)$, respectively, where M is the number of tasks in the ready queue and N is the processor-supported frequency levels. The overall time complexity of the proposed state-aware frequency selection algorithm is $O(MN)$. The time complexity of the benchmarking algorithm presented in [17] is $O(M^2N)$.

The number of elementary operations of the two algorithms is further investigated for the time complexity analysis. Let T_{proposed} denote the number of elementary operations of the proposed scheme, and let T_{hadvfs} denote the number of elementary operations of the benchmarking algorithm HA-DVFS. The T_{proposed} is then given by

$$T_{\text{proposed}} \leq M(N+1) + 5N + 17 \tag{15}$$

and T_{hadvfs} is given by

$$T_{\text{hadvfs}} \leq \frac{(M^2 + 17M)N}{4} + \log_{d_{\max}}^{M} + 6M + 8 \tag{16}$$

where M is the length of the ready queue, N is the processor-supported frequency levels, and $d_{\max}$ is the maximum relative deadline of tasks in the ready queue.

Fig. 4 shows the log–log plot of the time complexity for the proposed algorithm and the benchmarking algorithm HA-DVFS. The plot is based on the number of elementary operations of the two algorithms running on a processor with 6 frequency levels. It has been shown in the figure that the proposed algorithm has much lower time complexity when compared with the benchmarking scheme, which is more suitable for dynamic scheduling.

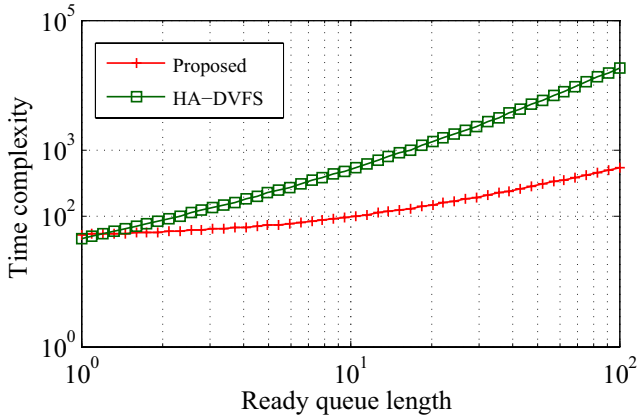


Fig. 4. Log–log plot of the time complexity.

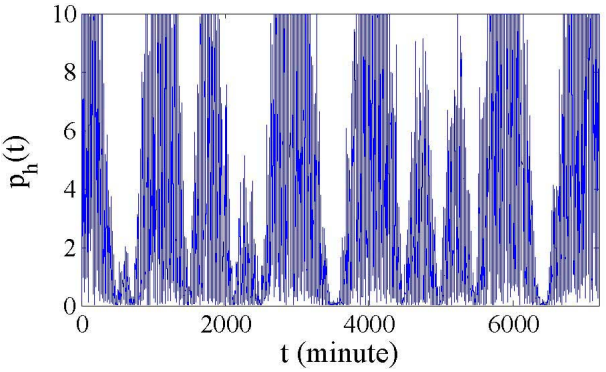


Fig. 5. Example of the obtained solar power trace.

TABLE I
XSCALE FREQUENCIES, SUPPLY VOLTAGES, AND POWER [25]

Frequency(MHz)	150	400	600	800	1000
Voltage(V)	0.75	1.0	1.3	1.6	1.8
Power(mW)	80	170	400	900	1600
Idle power(mW)	45				

The kernel part of the algorithm is the frequency selection scheme based on system utilization and energy state. Unlike the existing work that decouples the energy and timing constraint [17], the proposed scheme jointly accounts for constraints arising from both the energy and time domain. This strategy is motivated by the fact that the energy and time constraint interleave, and a joint scheduling approach will further improve system performance, as is demonstrated in Section V.

V. EXPERIMENTAL RESULTS AND DISCUSSIONS

Extensive simulation experiment has been performed to validate the proposed scheme in energy efficiency and deadline miss rate. The proposed scheme is implemented in C++ and run on a laptop equipped with an Intel Core i5 processor of 2.40 GHz, a 1 GB RAM, and the Ubuntu 10.04 operating system. The proposed algorithm is compared with benchmarking algorithms including the LSA [12] and HA-DVFS [17]. The LSA proposed by Moser *et al.* postpones the execution of a task as late as possible and the DVS technique is not employed to scale down the operating frequency. In other words, the LSA algorithm operates at a fixed operating frequency. The LSA algorithm is implemented in this paper for comparison study and it is assumed that the algorithm operates at the maximum processor-supported frequency. Unlike the LSA algorithm, the HA-DVFS algorithm proposed in [17] employs the DVS technique to scale the processor to the lowest possible frequency, and increases the operating frequency to enhance energy efficiency when overflow occurs. For the sake of fair comparison, the simulation settings adopted for the experiment are similar to those adopted in benchmarking schemes.

An Intel XScale processor supporting five frequency levels was utilized to estimate the energy consumption. The discrete frequencies, supply voltage, TDP power, and idle power of the processor are listed in Table I [25]. The solar energy is selected as the renewable generation in the experiment. The trace of harvesting power $p_h(t)$ is generated according to

$$p_h(t) = \left| 10 \cdot N(t) \cdot \cos\left(\frac{t}{70\pi}\right) \cdot \cos\left(\frac{t}{100\pi}\right) \right| \qquad (17)$$

where $N(t)$ is a random variable with variance of 1 and mean of 0 [12]. Fig. 5 shows the obtained power trace in a frame of about 7000 time units, which is about five days long. The average harvesting power $\overline{p_h}$ of the system is derived from this power trace.

The designed sets of real-time tasks are composed of arbitrary number of periodic and aperiodic tasks, and each task in a task set is generated based on the average harvesting power. The period of a periodic task is randomly drawn from the set $\{10, 20, 30, \ldots, 120\}$ under the assumption that each value in the set has the same probability of being selected. The relative deadline of a periodic task is assumed to equal the period of the task. This approach is also utilized to generate the relative deadlines of aperiodic tasks. Similarly, the release time of an aperiodic task is randomly generated in the simulation duration based on the uniform distribution of the probability. The energy of a task τ_m, which is denoted by e_m, is generated based on the uniform distribution of the probability in the interval of $[0, \overline{p_h} \times d_m]$, where $\overline{p_h}$ is the average harvesting power and d_m is the relative deadline of the task τ_m. The worst case execution time of the task is given by e_m/p_N, where p_N is the power consumption of the processor running at the maximum frequency f_N.

A. Comparison of the Deadline Miss Rate for Fixed Battery Capacity

The proposed task-scheduling algorithm is compared with the HA-DVFS algorithm in deadline miss rate. The battery capacity is assumed to be 1000 J and the initial energy stored in the battery is assumed to be 500 J. The system utilization, which is calculated as the sum of the utilizations of all tasks in the task set, takes the values of 0.2, 0.4, 0.6, and 0.8. 3000 task sets are generated for each value of the system utilization. Real-time tasks in each task set are scheduled using one of the three scheduling schemes and the simulation is run for 12 h. The ratio of the number of tasks that miss deadlines to the total number of tasks in a task set is defined to be the deadline miss

TABLE II
COMPARE THE PROPOSED ALGORITHM WITH HA-DVFS [17] WITH RESPECT TO DEADLINE MISS RATE

Utilization		Deadline miss rate (%)	
		HA-DVFS	Proposed
Periodic tasks	0.2	0.003	0
	0.4	0.40	0.39
	0.6	7.41	7.30
	0.8	19.81	18.58
Aperiodic tasks	0.2	0.01	0
	0.4	1.37	1.30
	0.6	10.07	10.02
	0.8	21.36	21.11

rate of the task set. The deadline miss rate given in Table II is the average value over the 3000 task sets.

Table II shows that the deadline miss rate of the proposed scheme is close to that of the HA-DVFS. Note that the time complexity of the proposed scheme is $O(MN)$ while the time complexity of the HA-DVFS is $O(M^2N)$, where M is the number of tasks in the task set and N is the processor-supported frequency levels. It also has been shown in the table that when the system utilization is low (e.g., 0.2), the deadline miss rate of the proposed scheme approaches 0 while the deadline miss rate of the HA-DVSF converges to a small constant value. This is because the HA-DVSF decouples the energy and time by always scaling the processor speed down to the lowest possible frequency even if the system utilization is small and large amount of energy is available, which may lead to task deadline violation.

B. Comparison of the Deadline Miss Rate for Varying Battery Capacity

The proposed scheme has been compared with the LSA [12] and HA-DVFS [17] algorithm with respect to deadline miss rate when the battery capacity varies. 3000 sets of real-time tasks are generated and simulation is performed for each task set for 12 h. The obtained deadline miss rate is averaged over the 3000 sets of tasks. Let U denote the system utilization.

Fig. 6 shows the comparison of the proposed scheme, LSA, and HA-DVFS algorithm when the battery capacity is small ($E_{\text{cap}} < 1000$ J). Since the LSA algorithm runs at the highest processor speed and does not utilize DVS technique to save energy, thus, it depletes the battery energy fast and incurs much higher deadline miss rate when compared with the proposed scheme and the HA-DVFS algorithm. The HA-DVFS is an aggressive scheme that always scales down the processor frequency to the lowest possible level while the proposed algorithm that jointly considers system utilization and energy state is more conservative. When the battery capacity is small the energy constraint becomes more critical, which is favorable for the HA-DVFS algorithm. As is shown in Fig. 6, the HA-DVFS outperforms the proposed algorithm for different values of utilization when the battery capacity is less than 30 J.

Fig. 7 shows the comparison of the proposed scheme, LSA, and HA-DVFS algorithm when the battery capacity is large ($E_{\text{cap}} > 1000$ J). When the battery capacity increases, the amount of energy in a full battery is enough to finish task execution. In this scenario, the energy constraint becomes less critical when compared with the timing constraint. As a result, the HA-DVFS algorithm that decouples the energy and timing constraint has higher deadline miss rate as compared with the proposed scheme that jointly considers the energy and time. For the case of $U = 0.6$, the proposed scheme outperforms the HA-DVFS in deadline miss rate when battery capacity is greater than 15000 J. For the case of $U = 0.8$, the deadline miss rate of the HA-DVFS exceeds that of the proposed scheduling algorithm by about 4.1% and 11.5% when battery capacity is greater than 5000 and 30 000 J, respectively, as is shown in Fig. 7(d).

With the increase in battery capacity, the deadline miss rate of the LSA algorithm decreases since there is enough energy to finish task execution. As is shown in Fig. 7(d), the

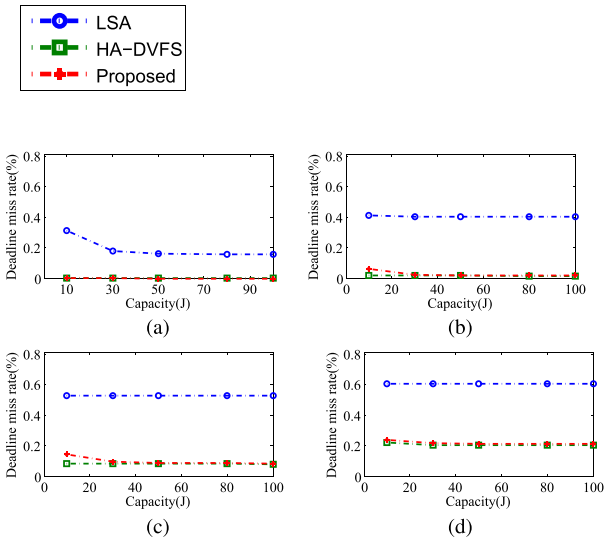


Fig. 6. (a)–(d) Compare the proposed scheme with LSA [12] and HA-DVFS [17] in deadline miss rate when battery capacity $E_{\text{cap}} < 1000$ J.

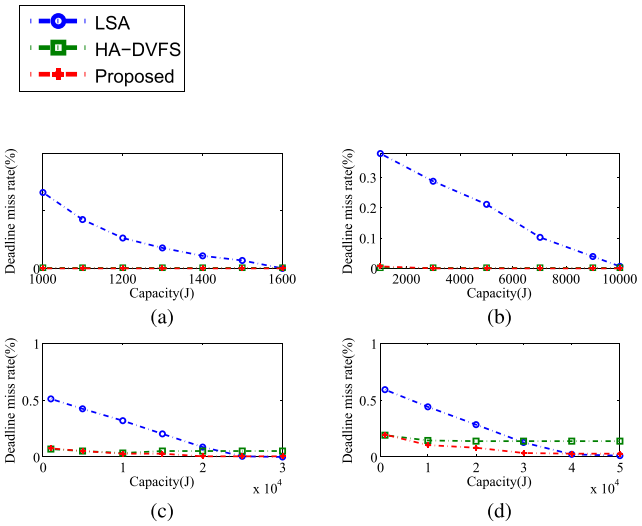


Fig. 7. (a)–(d) Compare the proposed scheme with LSA [12] and HA-DVFS [17] in deadline miss rate when battery capacity $E_{\text{cap}} > 1000$ J.

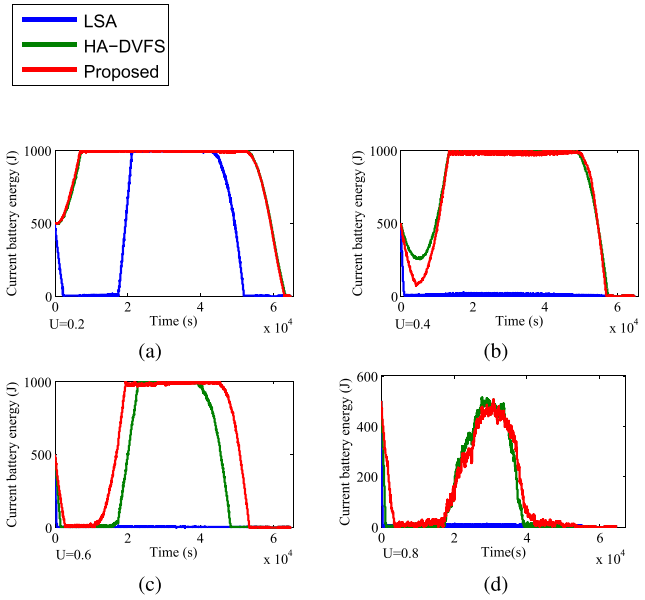


Fig. 8. (a)–(d) Amount of the current battery energy from 7:00 A.M. to 1:00 A.M. of the next day for system utilization $U = 0.2$, 0,4, 0.6, and 0.8.

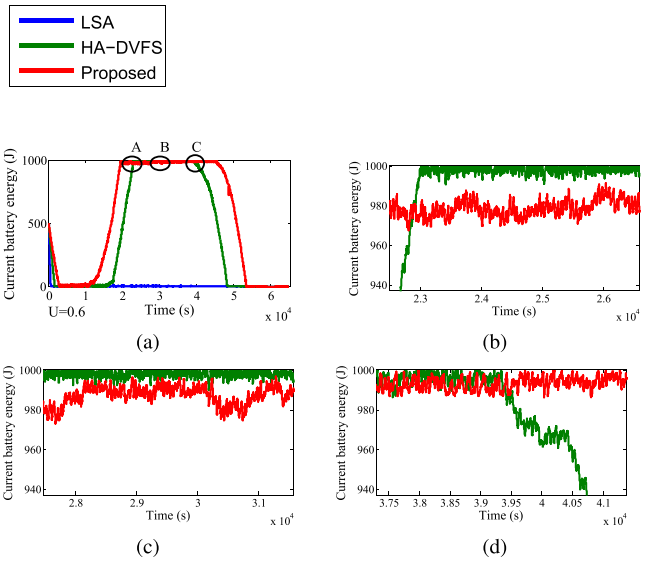


Fig. 9. (a)–(d) Snapshot of the current battery energy in areas A, B, and C for utilization $U = 0.6$.

LSA algorithm outperforms the HA-DVFS and the proposed scheme when the battery capacity approaches 30 000 and 40 000 J, respectively, which is the typical capacity of two to three pieces of zinc–carbon AA battery.

C. Comparison of Current Energy in the Battery

The battery capacity is crucial to system performance of an energy-harvesting system. For given battery capacity, it is desirable that the wasted energy due to overflow is minimal and the current amount of energy in the battery is large enough to maintain the normal operation of the system even in the worst case of energy-harvesting condition. Thus the current amount of energy in the battery is an important indicator of system performance.

It is assumed that the battery capacity is 1000 J and the initial energy stored in the battery is 500 J. The system utilization takes the values of $U = 0.2$, 0,4, 0.6, and 0.8, and simulation is performed for each value of the utilization. The current amount of energy in the battery is averaged over 3000 generated sets of real-time tasks, and the simulation for each task sets is run from 7:00 A.M. to 1:00 A.M. of the next day, which lasts for 18 h.

Fig. 8 shows that the amount of current energy in the battery varies with different scheduling algorithms. Of the three algorithms, the LSA has the smallest amount of current energy as it operates at the highest processor-supported frequency and does not employs DVS to save energy. Both the proposed scheme and the HA-DVFS store the excess solar energy in the day and save energy for perpetual operation in the night.

For the case of $U = 0.2$, the system workload is so low that the timing constraints of real-time tasks can be satisfied even at the lowest frequency level. Since the system does not switch to the overloaded state and the harvesting rate is low in this scenario, the proposed scheme is essentially reduced to the HA-DVFS algorithm, thus, the current battery energy is close under the two scheduling policies. For the case of $U = 0.4$, the HA-DVFS algorithm that scales down the processor to the lowest possible frequency tries to minimize the amount of battery energy to be used, however, the proposed scheme that jointly considers the energy and time may allocate a portion of battery energy to mitigate the overloaded state. As a result, the battery energy of the proposed scheme is less than that of the HA-DVFS before the system adapts to the harvesting power rate, as is shown in Fig. 8(b). For the case of $U = 0.8$, the system workload is heavy and both algorithms run at a relatively high processor speed without waste of the harvested energy, thus, the curve for the current battery energy is close.

Fig. 8(c) ($U = 0.6$) shows that when compared with the HA-DVFS, the proposed algorithm is more conservative when the harvesting rate is low ($0 \sim 10^4$ s), builds up the battery energy earlier when the harvesting rate increases ($10^4 \sim 2 \times 10^4$ s), maintains a near full battery energy for a longer period of time when the harvesting rate is high ($2 \times 10^4 \sim 4.5 \times 10^4$ s), and depletes the battery energy later when the harvesting rate decreases ($4.5 \times 10^4 \sim 6 \times 10^4$ s). The proposed algorithm wastes less energy via its adaptive characteristics when the harvesting rate is high. As a result, it finishes the execution of more workload when the harvesting power is high, which leaves more space to scale down the processor speed when the harvesting power is low. This property of the proposed algorithm contributes strikingly to the lower deadline miss rate of the algorithm.

Fig. 9 gives snapshots of the Fig. 8(c). The areas A, B, and C for the curve $U = 0.6$ are enlarged and details are shown in the figure. It has been shown in the figure that when the battery is near a full charge, both the proposed scheme and the HA-DVFS attempt to enhance energy efficiency by increasing processor speed. However, the HA-DVFS exceeds the battery capacity of $E_{\text{cap}} = 1000$ J more frequently than

TABLE III
COMPARE THE PROPOSED ALGORITHM AND THE HA-DVFS ALGORITHM [17] IN SCHEDULING OVERHEADS

Task set size	Proposed			HA-DVFS		
	$\overline{M}$	$\overline{T}$	$\overline{I}(\times 10^7)$	$\overline{M}$	$\overline{T}$	$\overline{I}(\times 10^7)$
10	9	110	4.86	8	454	5.93
20	19	180	7.74	18	1147	13.69
40	37	306	12.65	35	3180	35.00
80	78	593	22.09	77	11696	133.19
100	97	726	26.28	95	17233	191.60

the proposed scheme does, which indicates that the HA-DVFS incurs more energy loss. This is also due to the different design principle of the two schemes. The HA-DVFS decouples the energy and time constraints, and allocates the available energy to the next single task to be executed. When excess solar energy is produced, a single task cannot consume all the excess energy. On the contrary, the proposed scheme jointly considers system utilization and energy state, and allocates the excess solar energy to multiple tasks in the feasible interval, which effectively reduces energy waste.

D. Scheduling Overheads Analysis

The runtime performance of the proposed scheme and the benchmarking algorithm HA-DVFS is evaluated using the SimpleScalar, an open source computer architecture simulator that is capable of providing cycle level estimation of C-based applications [26]. The scheduling overheads of the two algorithms are examined from the perspectives of the queue length, the number of elementary operations, and the number of SimpleScalar simulation-based instructions.

Five types of real-time task sets are generated, each of which contains 10, 20, 40, 80, and 100 tasks, respectively. 1000 task set instances are generated for each type, the scheduling of real-time tasks in each instance is simulated for 3600 s, and all the reported results are averaged over the 1000 task set instances.

Let $\overline{M}$ denote the average queue length, let $\overline{T}$ denote the average number of elementary operations for scheduling a single task, and let $\overline{I}$ denote the average number of SimpleScalar simulation-based instructions for scheduling tasks in a task set. Note that the $\overline{M}$ and $\overline{I}$ are generated using SimpleScalar tool set while the $\overline{T}$ is calculated based on the generated $\overline{M}$ using (15) and (16). In fact, the $\overline{T}$ derived using the two equations can be deemed as the theoretically computed scheduling overheads while the $\overline{I}$ can be considered the practical scheduling overheads.

Table III compares the proposed scheme and the benchmarking algorithm HA-DVFS in scheduling overheads. It has been shown in the table that for task sets of varying sizes, the average queue length $\overline{M}$ of the two algorithms are close; however, the HA-DVFS has more computed scheduling overheads $\overline{T}$ and more practical scheduling overheads $\overline{I}$ when compared with the proposed scheme. For a task set of 100 real-time tasks, the average queue length of the two schemes is about 95 while the computed and the practical scheduling overheads of the HA-DVFS are about 24 and 7 times those of the proposed scheme.

It also has been shown in the table that the computed scheduling overheads $\overline{T}$ of the two algorithms conform well with the practical scheduling overheads $\overline{I}$. when the task set size increases from 10 to 100, the computed $\overline{T}$ and the practical $\overline{I}$ of the proposed algorithm increase by about 6.6 and 5.4 times, respectively, and the computed $\overline{T}$ and the practical $\overline{I}$ of the HA-DVFS increase by about 37.9 and 32.3 times, respectively. This result is consistent with the time complexity of the proposed and the HA-DVFS algorithms, which is $O(MN)$ and $O(M^2N)$, respectively, where M is the queue length and N is the processor-supported frequency levels.

VI. CONCLUSION

This paper proposes a dynamic frequency selection scheme for energy-harvesting real-time systems. The proposed scheme is featured by system state characterization from perspectives of the system utilization and harvested energy with respect to a certain interval. It maintains a queue for ready tasks. When a new task arrives, it updates the ready task queue by inserting the new task into the queue according to the EDF scheduling policy, performs the state-aware frequency selection procedure for tasks in the ready queue, and executes the task at the head of the queue at the derived frequency. A task is removed from the ready queue when its execution is finished. Simulation results demonstrate the effectiveness of the proposed scheme. Compared with the state-of-the-art scheme HA-DVFS, the proposed scheme achieves comparable deadline miss rate for varying values of system utilization at one order of magnitude lower time complexity. When the battery capacity is greater than 30 000 J, the deadline miss rate of the proposed scheme is about 11.5% lower than that of the HA-DVFS. The results also demonstrate that the proposed scheme achieves higher energy efficiency and is more resilient to the intermittent nature of solar energy.

When the battery is near a full charge or overflow occurs, the HA-DVFS exceeds the battery capacity more frequently than the proposed scheme does, indicating more energy waste. The future work seeks to investigate the dynamic task allocation and frequency selection scheme for multiprocessor systems based on the scheduling scheme proposed in this paper.

REFERENCES

[1] K. Lampka, K. Huang, and J. Chen, "Dynamic counters and the efficient and effective online power management of embedded real-time systems," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synthesis*, 2011, pp. 267–276.

[2] T. Esram and P. Chapman, "Comparison of photovoltaic array maximum power point tracking techniques," *IEEE Trans. Energy Convers.*, vol. 22, no. 2, pp. 439–449, Jun. 2007.

[3] K. Kobayashi, I. Takano, and Y. Sawada, "A study on a two stage maximum power point tracking control of a photovoltaic system under partially shaded insolation conditions," in *Proc. IEEE Power Eng. Soc. General Meeting*, Jul. 2003.

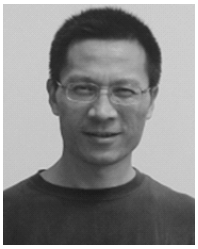
[4] K. Irisawa, T. Saito, I. Takano, and Y. Sawada, "Maximum power point tracking control of photovoltaic generation system under non-uniform insolation by means of monitoring cells," in *Proc. 28th IEEE Conf. Rec. Photovolt. Specialists Conf.*, Sep. 2000, pp. 1707–1710.

[5] M. Bodur and M. Ermis, "Maximum power point tracking for low power photovoltaic solar panels," in *Proc. 7th Medit. Electrotech. Conf.*, Apr. 1994, pp. 758–761.

[6] E. Lorenz, J. Hurka, D. Heinemann, and H. Beyer, "Irradiance forecasting for the power prediction of grid-connected photovoltaic systems," *IEEE J. Sel. Topics Appl. Earth Observat. Remote Sens.*, vol. 2, no. 3, pp. 2–10, Mar. 2009.
[7] A. Kansal, D. Potter, and M. Srivastava, "Performance aware tasking for environmentally powered sensor networks," in *Proc. Int. Conf. Meas. Model. Comput. Syst.*, 2004, pp. 223–234.
[8] V. Sharma, U. Mukherji, V. Joseph, and S. Gupta, "Optimal energy management policies for energy harvesting sensor nodes," *IEEE Trans. Wireless Commun.*, vol. 9, no. 4, pp. 1326–1336, Apr. 2010.
[9] Y. Abdeddaim and D. Masson, "Real-time scheduling of energy harvesting embedded systems with timed automata," in *Proc. Int. Conf. Embedded RTCSA*, 2012, pp. 31–40.
[10] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks," in *Proc. Int. Symp. Inf. Process. Sensor Netw.*, 2005, pp. 463–468.
[11] J. Hsu, A. Kansal, J. Friedman, V. Raghunathan, and M. Srivastava, "Energy harvesting support for sensor networks," in *Proc. Int. Symp. Inf. Process. Sensor Netw.*, 2005.
[12] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling for energy harvesting sensor nodes," *Real-Time Syst. J.*, vol. 37, no. 3, pp. 233–260, Dec. 2007.
[13] C. Moser, J. Chen, and L. Thiele, "An energy management framework for energy harvesting embedded systems," *ACM J. Emerging Technol. Comput. Syst.*, vol. 6, no. 2, pp. 1–7, Jun. 2010.
[14] C. Moser, L. Thiele, D. Brunelli, and L. Benini, "Adaptive power management for environmentally powered systems," *IEEE Trans. Comput.*, vol. 59, no. 4, pp. 478–491, Apr. 2010.
[15] H. Ghor, M. Chetto, and R. Chehade, "A real-time scheduling framework for embedded systems with environmental energy harvesting," *J. Comput. Electr. Eng.*, vol. 37, no. 4, pp. 498–510, Jul. 2011.
[16] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling with regenerative energy," in *Proc. 18th Euromicro Conf. Real-Time Syst.*, 2006, pp. 1–10.
[17] S. Liu, Q. Qiu, and Q. Wu, "Harvesting-aware power management for real-time systems with renewable energy," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 8, pp. 1473–1486, Aug. 2011.
[18] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A System Perspective*. Reading, MA, USA: Addison-Wesley, 1992.
[19] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proc. 41st DAC*, Jul. 2004, pp. 275–280.
[20] L. Martín, L. Zarzalejo, J. Polo, A. Navarro, R. Marchante, and M. Cony, "Prediction of global solar irradiance based on time series analysis: Application to solar thermal power plants energy production planning," *Solar Energy Elsevier*, vol. 84, no. 10, pp. 1772–1781, Oct. 2010.
[21] T. Khatib, A. Mohamed, and K. Sopian, "A review of solar energy modeling techniques," *Renew. Sustain. Energy Rev. Elsevier*, vol. 16, no. 5, pp. 2864–2869, 2012.
[22] D. Fadare, "Modelling of solar energy potential in Nigeria using an artificial neural network model," *Appl. Energy Elsevier*, vol. 86, no. 9, pp. 1410–1422, 2009.
[23] A. Mellit and A. Pavan, "A 24-h forecast of solar irradiance using artificial neural network: Application for performance prediction of a grid-connected PV plant at Trieste, Italy," *Solar Energy Elsevier*, vol. 84, no. 5, pp. 807–821, May 2010.
[24] A. Palit and D. Popovic, *Computational Intelligence in Time Series Forecasting*. New York, NY, USA: Springer-Verlag, 2005.
[25] Intel Corp. (2004). *Intel XScale Processor Family Electrical, Mechanical, and Thermal Specification Datasheet*, Santa Clara, CA, USA [Online]. Available: http://developer.intel.com
[26] (2010). *SimpleScalar Computer Architecture Simulator* [Online]. Available: http://www.simplescalar.com

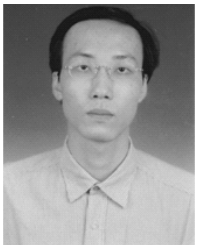
Jing Chen is currently pursuing the master's degree with the Department of Computer Science and Technology, East China Normal University, Shanghai, China.

Her current research interests include the management of energy and reliability for real-time embedded systems.



Tongquan Wei (M'11) received the B.S. degree in electronics engineering from the Dalian University of Technology, Dalian, China, in 1995, the M.S. degree in computer engineering from the University of Missouri-Rolla (now Missouri University of Science and Technology), Rolla, MO, USA, and the Ph.D. degree in electrical engineering from Michigan Technological University, Houghton, MI, USA, in 2003 and 2009, respectively.

He has been with East China Normal University, Shanghai, China, since 2009, where he is currently an Associate Professor with the Department of Computer Science and Technology. His current research interests include real-time systems, low power, fault tolerance, and hardware-software co-design.



Jianlin Liang received the M.S. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 2002.

He was with Broadcom Corporation, San Diego, CA, USA, from 2007 to 2011, on Bluetooth related digital ASIC backend design. His current research interests include digital ASIC design and embedded systems.