Des Autom Embed Syst (2007) 11:25–48
DOI 10.1007/s10617-006-9724-2

Reward-based voltage scheduling for dynamic-priority hard real-time systems

Han-Saem Yun · Jihong Kim

Abstract *Reward-based scheduling* has been investigated for flexible applications in which an approximate but timely result is acceptable. Meanwhile, significant research efforts have been made on *voltage scheduling* which exploits the tradeoff between the processor speed and the energy consumption. In this paper, we address the combined scheduling problem of maximizing the total reward of hard real-time systems with a given energy budget. We present an *optimal off-line* algorithm and an efficient *on-line* algorithm for jobs with their own release-times/deadlines under Earliest-Deadline-First (EDF) scheduling. Experimental results show that the solution computed by the on-line algorithm is no more than 14% worse than the theoretical optimal solution obtained by the optimal off-line algorithm.

Keywords Dynamic voltage scaling · Variable voltage processor · Reward-based scheduling

1 Introduction

Reward-based scheduling [2] has been introduced to handle overloaded real-time systems, for which it is not possible to meet all the timing constraints unless some tasks are allowed to be skipped entirely or executed partially. In the reward-based scheduling framework, the workload of each task is divided into a mandatory part and an optional part. The mandatory part of a task should be completed by its deadline while the optional part can be selectively

This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment).

A preliminary version of this article was presented at *Real-Time and Embedded Computing Systems and Applications (RTCSA'04).*

H.-S. Yun · J. Kim (✉)
School of Computer Science and Engineering, Seoul National University, Seoul, Korea 151-742
e-mail: jihong@davinci.snu.ac.kr

H.-S. Yun
e-mail: hsyun@davinci.snu.ac.kr

executed before the deadline. The optional part is assumed to follow the mandatory part in sequence and can be interrupted at any time. A reward function is associated with each optional part; the more the optional part is executed, the higher the reward is. The reward-based framework can model various real-time applications that allow approximate results such as image and speech processing, multimedia, robot control/navigation systems, information gathering, real-time heuristic search [2]. We call these applications *flexible applications* [13]. The goal of reward-based scheduling is to find optional parts that maximizes the total reward while meeting all the deadlines of the tasks composed of the fixed mandatory parts and the optional parts computed.

Recently, the energy consumption has been one of the most important design constraints, especially for mobile devices that operate with a limited energy source such as batteries. Because the dynamic energy consumption, which dominates the total energy of CMOS circuits, is quadratically dependent on the supply voltage, lowering the supply voltage is effective in reducing the energy consumption. However, lowering the supply voltage also decreases the clock speed [17]. When a given application does not require the peak performance of a VLSI system, in order to save the energy, the clock speed (and its corresponding supply voltage) can be dynamically adjusted to the lowest level that still satisfies the required performance. This is the key principle of *voltage scheduling* technique. With a recent explosive growth of the portable embedded system market, several commercial variable-voltage processors were developed (e.g, Intel's *Xscale*, AMD's *K6-2+*, and Transmeta's *Crusoe* processors.) Targeting these processors, various voltage scheduling algorithms have been developed. The goal of voltage scheduling is to find an energy-efficient voltage schedule with all the stringent timing constraints satisfied. A voltage schedule is a function that associates each time unit with a voltage level (i.e., a clock frequency).

As flexible applications are executed on variable voltage processors, the combined problem of reward-based scheduling and voltage scheduling, which we call the *reward-based voltage scheduling* problem, has been investigated [15, 16]. The reward-based voltage scheduling problem can be viewed as a generalized version of either reward-based scheduling or voltage scheduling by adding one more dimension to the solution space of these problems; for the former, the processor speed as a function of time is additionally computed along with the optional workloads, while for the latter the optional workload of each task is determined along with the voltage schedule.

The reward-based voltage scheduling problem involves two-dimensional objectives, maximizing the total reward (from reward-based scheduling) and minimizing the energy consumption (from voltage scheduling), and can be defined as duals; maximizing the total reward within a given energy budget or minimizing the energy consumption while providing the acceptable quality defined by reward functions. By considering different values of the constraint and solving the corresponding problem, designers can obtain Pareto-optimal points which represent the exact trade-off between the solution quality and the amount of energy required. Without loss of generality, in this paper, we consider the problem of maximizing the total reward subject to energy constraints.

1.1 Previous work

Reward-based execution model [2] has its origin in the *IC* (*Imprecise Computation*) [4,12] and *IRIS* (*Increasing Reward with Increasing Service*) [5] models. In the IC model, an optional part is associated with a *decreasing linear* function that indicates the precision error, and the goal is to minimize the weighted sum of the errors. Several optimal off-line algorithms have been proposed for aperiodic IC tasks [18]. Note that an IC model can be transformed into a reward-based model by substituting increasing linear reward functions for the decreasing error functions. The IRIS model corresponds to the special case of the reward-based model without mandatory parts. In [5], an optimal off-line algorithm and an on-line algorithm for the IRIS aperiodic tasks are presented.

Aydin et al. proposed the generalized reward-based execution model and developed an optimal off-line algorithm for periodic tasks with concave reward functions [2]. Concave functions (including linear functions) can model the output quality of several flexible applications such as multimedia applications, real-time heuristic search, pattern recognition, and database query processing [2]. They also proved that the problem for convex reward functions is NP-hard [2]. For firm real-time applications, of which reward is given by step functions, the reward-based scheduling problem is NP-complete [18].

Voltage scheduling for variable-voltage processors has recently been extensively studied targeting various system models. Voltage scheduling algorithms are classified into off-line and on-line algorithms. Off-line algorithms compute static voltage schedules with the assumption that timing parameters of each job is constant and known a priori while on-line algorithms dynamically adjust the processor speed along with the supply voltage based on the execution history.

For static job sets where each job has its own release time, deadline, and workload known offline,[1] Yao et al. proposed an optimal off-line voltage scheduling algorithm assuming EDF scheduling policy [20]. The off-line scheduling problem for the static job model with arbitrary priority assignment (including RM (Rate-Monotonic) or DM (deadline-monotonic) assignment) was proved to be NP-hard, and a fully polynomial time approximation scheme (FPTAS) for the problem was presented [22]. Several on-line voltage scheduling algorithms have been developed for both EDF periodic tasks [1, 7, 8, 14] and fixed-priority periodic tasks [6, 9, 14, 19]. Quantitative evaluation of existing on-line algorithms are presented in [10].

Reward-based voltage scheduling was first addressed by Rusu et al. [15, 16]. In [16], off-line solutions for frame-based task sets (where all the jobs have *identical* release times and deadlines) and periodic EDF task sets with concave reward functions are considered. They showed that the problem for periodic EDF task sets can be reduced to the problem for the frame-based task sets. For tasks with identical power functions (i.e., the same switching activity), they proved that all the tasks run at the same speed under the optimal schedule. Thus, the problem is simply reduced to the reward-based scheduling problem solved in [2]. They also developed an efficient off-line heuristic for tasks with different power functions. The reward-based voltage scheduling problem for frame-based task sets with 0/1 reward functions (i.e., no reward is given unless the optional part is completely executed.) is proved to be NP-hard and a heuristic for the problem is presented in [15]. The reward-based voltage scheduling remains relatively unexplored partly due to the complexity caused by multidimensional solution space (i.e., one dimension from voltage scheduling and the other from reward-based scheduling).

1.2 Contributions

In this paper, we consider reward-based voltage scheduling for the *general* task model used in [20, 22] unlike the restricted task model (e.g., frame-based task sets used in the previous work [15, 16]). First, we describe an *optimal off-line* algorithm under the assumption that

[1] Note that the typical periodic task model can be transformed into the static job model by considering all the task instances within a hyperperiod of periodic tasks.

the amount of workload (i.e., mandatory part and optional part) of each job is fixed and known a priori. Second, we present an efficient *on-line* algorithm which effectively leverages workload variations to increase the reward within energy budget. Experimental results show that the on-line algorithm is sufficiently efficient; the quality of solution (i.e., the total reward) computed by the on-line algorithm is no more than 14% worse than that of the theoretical optimal solution obtained by the optimal off-line algorithm.

The rest of the paper is organized as follows. We formulate the problem in Section 2. The optimal off-line algorithm is described in Sections 3. The on-line algorithm is presented in Section 4. In Section 5, the experimental results are discussed. Section 6 concludes with a summary and directions for future work.

2 Problem formulation

We consider a set $\mathcal{J} = \{J_1, J_2, \ldots, J_{|\mathcal{J}|}\}$ of priority-ordered jobs with J_1 being the job with the highest priority. A job $J_i \in \mathcal{J}$ is associated with the following attributes, which are assumed to be known off-line:

- r_i and d_i: the release time and the deadline.
- m_i: the mandatory workload expressed in execution cycles.
- u_i: the sum of m_i and the upper bound of the optional workload. (i.e., the optional workload (resp. the total workload) is selected between $[0, u_i - m_i]$ (resp. $[m_i, u_i]$).)
- $\rho_i(\cdot)$: the reward given as a function of the total workload.

We assume that the job set $\mathcal{J}$ follows the EDF priority as in [20]. (A job set $\mathcal{J}$ is said to be an EDF job set if for any $1 \le i < j \le |\mathcal{J}|$, $d_i \le d_j$ or $d_j \le r_i$.) A total order on the priorities can be provided by preferring the job with the earliest release time. For the on-line scheduling problem, m_i and u_i are the worst-case values and the actual mandatory workload and upper bound of the optional workload vary within $(0, m_i]$ and $(0, u_i - m_i]$ during runtime.

The total workload of J_i (i.e., the sum of the mandatory and optional workloads of J_i) is denoted by o_i and is selected between $[m_i, u_i]$, i.e., $m_i \le o_i \le u_i$. From now on, we call o_i and $\mathbf{o} = (o_1, o_2, \ldots, o_{|\mathcal{J}|})$ the workload of J_i and the workload tuple, respectively. Associated with each optional workload o_i is a reward function $\rho_i(o_i)$, which is assume to be strictly increasing, concave, and continuously differentiable over the interval $[m_i, u_i]$ as in [2, 16]. The derivative of $\rho_i(\cdot)$ is denoted by $\rho_i'(\cdot)$.

Given a workload tuple $\mathbf{o} = (o_1, o_2, \ldots, o_{|\mathcal{J}|})$, the total reward F, our optimization goal, is given by $F(\mathbf{o}) = \sum_{i=1}^{|\mathcal{J}|} \rho_i(o_i)$. For a given workload tuple $\mathbf{o}$, the workload of J_i is addressed by $o_i[\mathbf{o}]$, or briefly o_i when no confusion arises. Particularly, we use $\mathbf{m}$ and $\mathbf{u}$ to denote $(m_1, m_2, \ldots, m_{|\mathcal{J}|})$ and $(u_1, u_2, \ldots, u_{|\mathcal{J}|})$, respectively. Note that $\mathbf{m}$ and $\mathbf{u}$ are the lower and upper bounds for $\mathbf{o}$, respectively. The solution space of $\mathbf{o}$ is written by $\mathcal{O}_{\mathcal{J}}$, i.e., $\mathcal{O}_{\mathcal{J}} = \{\mathbf{o} | \mathbf{m} \le \mathbf{o} \le \mathbf{u}\}$.

Since there is a one-to-one correspondence between the processor speed and the supply voltage, we use $\mathcal{S}(t)$, the processor speed, to denote the voltage schedule. Given a workload tuple $\mathbf{o}$, a voltage schedule $\mathcal{S}(t)$ is said to be *feasible* for $\mathbf{o}$ if $\mathcal{S}(t)$ gives each job J_i the required number of cycles $o_i[\mathbf{o}]$ between its release time r_i and deadline d_i. As with other related work [20, 22], we assume that the processor speed can be varied continuously[2] with

[2] In [21], we extend the results to a processor model with a limited number of voltage levels using the simple transformation between the continuous-voltage version problem and the discrete-voltage version problem proposed by Kwon and Kim [11].

a negligible overhead both in time and power. Furthermore, we model that the power $P(\cdot)$, energy consumed per unit time, is a convex function of the processor speed; given a voltage schedule $\mathcal{S}(t)$, the power can be written as a function of time by $P(\mathcal{S}(t))$. For simplicity, we assume that all the jobs have the same switching activity and that $P(\cdot)$ is dependent only on the processor speed.

The energy-optimal voltage schedule for $\mathbf{o}$ is a voltage schedule $\mathcal{S}(t)$ feasible for $\mathbf{o}$ that minimizes $\int_{t_s}^{t_f} P(\mathcal{S}(t))dt$ where t_s and t_f are the lower and upper limits of release times and deadlines of the jobs in $\mathcal{J}$, respectively. The energy-optimal voltage schedule, written as $\mathcal{S}[\mathbf{o}]$, is unique and can be obtained by Yao's algorithm [20] in polynomial time.

From the fact that each job runs at the constant speed under an energy-optimal voltage schedule [20, 22], we can easily establish a one-to-one correspondence between $\mathcal{S}(t)$ and the the allowed execution time a_i allocated to each $J_i \in \mathcal{J}$ for a fixed workload tuple $\mathbf{o}$. Given a feasible voltage schedule $\mathcal{S}$, the corresponding tuple $(a_1, a_2, \ldots, a_{|\mathcal{J}|})$ of the allowed execution times, called a *time-allocation tuple*, is uniquely determined. Conversely, given a time-allocation tuple $\mathbf{A} = (a_1, a_2, \ldots, a_{|\mathcal{J}|})$, the corresponding voltage schedule $\mathcal{S}_\mathbf{A}$ can be uniquely constructed by assigning the constant execution speed o_i/a_i to J_i.

The solution space of reward-based voltage scheduling consists of the workload tuple $\mathbf{o}$ (from reward-based scheduling) and the voltage schedule $\mathcal{S}$ (from voltage scheduling). Note that the response time of a job J_i is uniquely determined by a time-allocation tuple $\mathbf{A}$ (regardless of $\mathbf{o}$) and the corresponding voltage schedule $\mathcal{S}_\mathbf{A}$ is feasible if and only if the response time of a job J_i is shorter than or equal to $d_i - r_i$ for all $1 \leq i \leq |\mathcal{J}|$. Thus, in order to simplify searching the solution space, we consider the time-allocation tuple $\mathbf{A}$ and the feasibility condition in terms of $\mathbf{A}$ instead of the voltage schedule $\mathcal{S}$. In the following, we define the optimization goal (i.e., the total reward) and the constraints (i.e., the feasibility condition and the energy constraints) in terms of the pair $\langle \mathbf{A}, \mathbf{o} \rangle$, called a *schedule*. For an EDF job set $\mathcal{J}$, $\mathbf{A} = (a_1, a_2, \ldots, a_{|\mathcal{J}|})$ is feasible if and only if the following condition is satisfied (See [22] for a proof.):

Condition I (Feasibility Condition).

$$\text{For any } r_i < d_j \ \ (1 \leq i\ ,\ j \ \leq |\mathcal{J}|)\ , \qquad \sum_{k/[r_k, d_k] \subseteq [r_i, d_j]} a_k \ \leq \ d_j - r_i \ .$$

For a given job set $\mathcal{J}$, the set of all feasible $\mathbf{A}$'s is denoted by $\mathcal{F}_\mathcal{J}$, i.e., $\mathcal{F}_\mathcal{J} = \{\mathbf{A} \mid \mathbf{A} = (a_1, \ldots, a_{|\mathcal{J}|})$ satisfies Condition I.$\}$. A schedule $\langle \mathbf{A}, \mathbf{o} \rangle$ is said to be feasible if $\mathbf{A}$ is feasible (i.e., $\mathbf{A} \in \mathcal{F}_\mathcal{J}$) and $\mathbf{o} \in \mathcal{O}_\mathcal{J}$ (i.e., $\mathbf{m} \leq \mathbf{o} \leq \mathbf{u}$). Now, we can specify the solution space of the problem as the cartesian product $\mathcal{F}_\mathcal{J} \times \mathcal{O}_\mathcal{J}$, which is the set of feasible schedules, i.e.,

$$\begin{aligned} \mathcal{F}_\mathcal{J} \times \mathcal{O}_\mathcal{J} &= \{\mathbf{A} \mid \mathbf{A} \text{ satisfies Condition I.}\} \times \{\mathbf{o} \mid \mathbf{m} \leq \mathbf{o} \leq \mathbf{u}\} \\ &= \{\langle \mathbf{A}, \mathbf{o} \rangle \mid \langle \mathbf{A}, \mathbf{o} \rangle \ \text{ is feasible.}\}. \end{aligned}$$

Figure 1 shows an example job set $\mathcal{J} = \{J_1, J_2, J_3, J_4, J_5\}$ where $(r_1, d_1) = (1, 5)$, $(r_2, d_2) = (4, 6)$, $(r_3, d_3) = (2, 7)$, $(r_4, d_4) = (9, 11)$ and $(r_4, d_4) = (0, 13)$. In Fig. 1(a), jobs are assigned the allowed execution times $\mathbf{A} = (a_1, a_2, a_3, a_4, a_5) = (2, 2, 2, 2, 5)$ and the workloads $\mathbf{o} = (o_1, o_2, o_3, o_4, o_5) = (4, 6, 4, 4, 5)$, respectively. Figure 1(b) shows an alternative schedule; $\langle \mathbf{A}, \mathbf{o} \rangle = \langle (2, 2, 2, 2, 5), (\sqrt[3]{102}, \sqrt[3]{102}, \sqrt[3]{102}, \sqrt[3]{102}, 5) \rangle$.

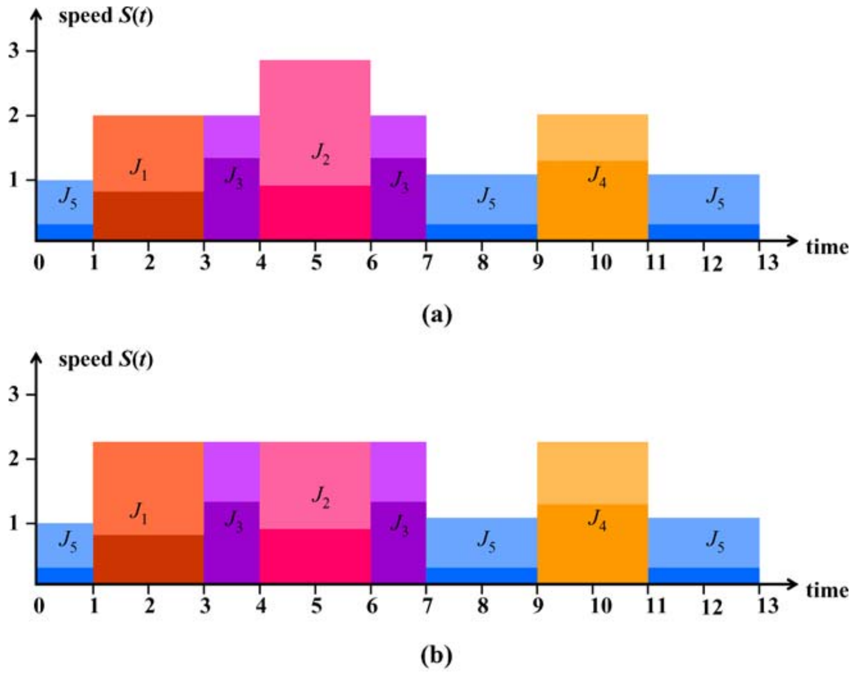

Fig. 1 Examples of schedules for the job set $\mathcal{J} = \{J_1, J_2, J_3, J_4, J_5\}$ where $(r_1, d_1) = (1, 5), (r_2, d_2) = (4, 6), (r_3, d_3) = (2, 7), (r_4, d_4) = (9, 11)$ and $(r_4, d_4) = (0, 13)$; (a) $\mathbf{A} = (a_1, a_2, a_3, a_4, a_5) = (2, 2, 2, 2, 5)$, $\mathbf{o} = (o_1, o_2, o_3, o_4, o_5) = (4, 6, 4, 4, 5)$ and (b) $\mathbf{A} = (2, 2, 2, 2, 5)$, $\mathbf{o} = (\sqrt[3]{102}, \sqrt[3]{102}, \sqrt[3]{102}, \sqrt[3]{102}, 5)$

Given a schedule $\langle \mathbf{A}, \mathbf{o} \rangle$, the energy consumption is given by $E(\langle \mathbf{A}, \mathbf{o} \rangle) = \sum_{i=1}^{|\mathcal{J}|} a_i \cdot P(o_i/a_i)$. Then, the reward-based voltage scheduling problem is formulated as follows:

Reward-Based Voltage Scheduling Problem
Find a schedule $\langle \mathbf{A}, \mathbf{o} \rangle \in \mathcal{F}_{\mathcal{J}} \times \mathcal{O}_{\mathcal{J}}$ that maximizes $F(\mathbf{o}) = \sum_{i=1}^{|\mathcal{J}|} \rho_i(o_i)$ subject to $E(\langle \mathbf{A}, \mathbf{o} \rangle) = \sum_{i=1}^{|\mathcal{J}|} a_i \cdot P(o_i/a_i) \leq E_{\text{budget}}$.

Assume that $P(s) = s^3$, $E_{\text{budget}} = 107$ and the reward functions are given by $\rho_i(x) = x$ for $1 \leq i \leq 5$. Then, the schedules in Fig. 1 satisfy both the timing contraints (i.e., Condition I) and the energy constraints. However, the total reward of the schedule in Fig. 1(b) is larger. (i.e., $4 \cdot \sqrt[3]{102} + 5 > 23$.)

For a fixed workload tuple $\mathbf{o}$, the energy-optimal voltage scheduling problem is stated as finding $\mathbf{A} \in \mathcal{F}_{\mathcal{J}}$ that minimizes $E(\langle \mathbf{A}, \mathbf{o} \rangle)$. We denote such $\mathbf{A}$ by $\mathbf{A}[\mathbf{o}] = (a_1[\mathbf{o}], a_2[\mathbf{o}], \ldots, a_{|\mathcal{J}|}[\mathbf{o}])$.[3] By using the parameterized expression $\mathbf{A}[\mathbf{o}]$, we can obtain an equivalent formulation in which the solution space is given only by $\mathcal{O}_{\mathcal{J}}$:

[3] In the rest of the paper, we use $\mathcal{S}[\mathbf{o}]$ and $\mathbf{A}[\mathbf{o}]$ interchangeably to denote the energy-optimal voltage schedule for $\mathbf{o}$.

$$\boxed{\begin{array}{l}\text{Find a workload tuple } \mathbf{o} \in \mathcal{O}_{\mathcal{J}} \text{ that maximizes } F(\mathbf{o}) \;=\; \sum_{i=1}^{|\mathcal{J}|} \rho_i(o_i) \\ \text{subject to } E(\langle \mathbf{A}[\mathbf{o}], \mathbf{o} \rangle) \;=\; \sum_{i=1}^{|\mathcal{J}|} a_i[\mathbf{o}] \cdot P(o_i/a_i[\mathbf{o}]) \;\leq\; E_{\text{budget}}.\end{array}}$$

The main source of difficulty is that $\mathbf{A}[\mathbf{o}]$ is not explicitly represented in terms of $\mathbf{o} = \{o_1, o_2, \ldots, o_{|\mathcal{J}|}\}$, thus making it difficult to explore the solution space implicitly given by a condition where $\mathbf{A}[\mathbf{o}]$ is involved (i.e., $E(\mathbf{A}[\mathbf{o}]) \leq E_{\text{budget}}$). In Section 2.1, we characterize some useful properties of energy-optimal voltage schedules, which provides a basis of an optimal off-line algorithm for the problem.

2.1 Characterization of energy-optimal voltage schedules

We first introduce notations which represent attributes of energy-optimal voltage schedules. In the following, notations are defined for an arbitrary but fixed workload tuple $\mathbf{o}$ and its corresponding energy-optimal voltage schedule $\mathbf{A}[\mathbf{o}]$ (equivalently, $\mathcal{S}[\mathbf{o}]$). $s_i[\mathbf{o}]$ and $I_i[\mathbf{o}]$ denote the (constant) speed of J_i and the union of intervals in which J_i is executed under $\mathbf{A}[\mathbf{o}]$, respectively, i.e., $\|I_i[\mathbf{o}]\| = a_i[\mathbf{o}]$ and $s_i[\mathbf{o}] = o_i[\mathbf{o}]/a_i[\mathbf{o}]$. We call $I_i[\mathbf{o}]$ the *execution interval* of J_i. $g_i[\mathbf{o}]$ is used to denote $P'(s_i[\mathbf{o}])/\rho_i'(o_i[\mathbf{o}])$ where P' is the derivative of P, and is called the *gradient* of J_i.

The voltage schedule in Fig. 1(a) is the energy-optimal voltage schedule for the workload tuple $\mathbf{o} = (4, 6, 4, 4, 5)$, i.e., $\mathbf{A}[\mathbf{o}] = (a_1[\mathbf{o}], a_2[\mathbf{o}], a_3[\mathbf{o}], a_4[\mathbf{o}], a_5[\mathbf{o}],) = (2, 2, 2, 2, 5)$. Under the voltage schedule $\mathbf{A}[\mathbf{o}]$, $s_1[\mathbf{o}] = s_3[\mathbf{o}] = s_4[\mathbf{o}] = 2$, $s_2[\mathbf{o}] = 3$ and $s_5[\mathbf{o}] = 1$; $I_1[\mathbf{o}] = [1, 3]$, $I_2[\mathbf{o}] = [4, 6]$, $I_3[\mathbf{o}] = [3, 4] \cup [6, 7]$ and $I_5[\mathbf{o}] = [0, 1] \cup [7, 9] \cup [11, 13]$; and $g_1[\mathbf{o}] = g_3[\mathbf{o}] = g_4[\mathbf{o}] = 12$, $g_2[\mathbf{o}] = 27$ and $g_5[\mathbf{o}] = 3$.

$\mathcal{J}_k[\mathbf{o}]$ represents the set of jobs scheduled at the k-th iteration of Yao's algorithm, and $\sigma_k[\mathbf{o}]$ denotes the constant speed allocated to jobs in $\mathcal{J}_k[\mathbf{o}]$. Note that $\sigma_k[\mathbf{o}]$ is nonincreasing with respect to k. $it_i[\mathbf{o}]$ is used to denote the iteration number k such that $J_i \in \mathcal{J}_k[\mathbf{o}]$, i.e., $J_i \in \mathcal{J}_{it_i[\mathbf{o}]}[\mathbf{o}]$. The union of execution intervals of jobs in $\mathcal{J}_k[\mathbf{o}]$ is denoted by $\mathbf{I}_k[\mathbf{o}]$, and called the *execution interval* of $\mathcal{J}_k[\mathbf{o}]$, i.e., $\mathbf{I}_k[\mathbf{o}] = \cup_{J_i \in \mathcal{J}_k[\mathbf{o}]} I_i[\mathbf{o}]$ and $\|\mathbf{I}_k[\mathbf{o}]\| = \sum_{J_i \in \mathcal{J}_k[\mathbf{o}]} a_i[\mathbf{o}]$. The set $\{\mathcal{J}_k[\mathbf{o}] \mid k = 1, 2, \ldots\}$ is a partition of $\mathcal{J}$, and is represented by $\mathcal{G}[\mathbf{o}]$. In Fig. 2, Yao's algorithm is described by the symbols defined so far.

For the example in Fig. 1(a), $\mathcal{J}_1[\mathbf{o}] = \{J_2\}$, $\mathcal{J}_2[\mathbf{o}] = \{J_1, J_3\}$, $\mathcal{J}_3[\mathbf{o}] = \{J_4\}$ and $\mathcal{J}_4[\mathbf{o}] = \{J_5\}$; $\sigma_1[\mathbf{o}] = 3$, $\sigma_2[\mathbf{o}] = \sigma_3[\mathbf{o}] = 2$, $\sigma_4[\mathbf{o}] = 1$; $it_1[\mathbf{o}] = it_3[\mathbf{o}] = 2$, $it_2[\mathbf{o}] = 1$, $it_4[\mathbf{o}] = 3$ and $it_5[\mathbf{o}] = 4$; $\mathbf{I}_1[\mathbf{o}] = [4, 6]$, $\mathbf{I}_2[\mathbf{o}] = [1, 4] \cup [6, 7]$, $\mathbf{I}_3[\mathbf{o}] = [9, 11]$ and $\mathbf{I}_4[\mathbf{o}] = [0, 1] \cup [7, 9] \cup [11, 13]$; $\mathcal{G}[\mathbf{o}] = \{\{J_2\}, \{J_1, J_3\}, \{J_4\}, \{J_5\}\}$.

$\mathcal{J}_k[\mathbf{o}]$ is partitioned into $\mathcal{J}_k^0[\mathbf{o}]$ and $\mathcal{J}_k^+[\mathbf{o}]$ such that a job $J_i \in \mathcal{J}_k[\mathbf{o}]$ belongs to $\mathcal{J}_k^0[\mathbf{o}]$ if $o_i[\mathbf{o}] = m_i$ and, otherwise, it belongs to $\mathcal{J}_k^+[\mathbf{o}]$, i.e.,

$$\mathcal{J}_k^0[\mathbf{o}] \overset{\text{def}}{=} \{J_i \in \mathcal{J}_k[\mathbf{o}] \mid o_i[\mathbf{o}] = m_i\} \quad \text{and} \quad \mathcal{J}_k^+[\mathbf{o}] \overset{\text{def}}{=} \{J_i \in \mathcal{J}_k[\mathbf{o}] \mid m_i < o_i[\mathbf{o}] \;\; (\leq u_i)\}.$$

For jobs in $\mathcal{J}_k^+[\mathbf{o}]$, the smallest ρ_i' value and the largest gradient are denoted by $\lambda_k[\mathbf{o}]$ and $\nabla_k[\mathbf{o}]$, respectively, i.e.,

$$\begin{aligned}\lambda_k[\mathbf{o}] &\overset{\text{def}}{=} \min\{\rho_i'(o_i[\mathbf{o}]) \mid J_i \in \mathcal{J}_k^+[\mathbf{o}]\} \quad \text{and} \quad \nabla_k[\mathbf{o}] \overset{\text{def}}{=} \max\{g_i[\mathbf{o}] \mid J_i \in \mathcal{J}_k^+[\mathbf{o}]\} \\ &\equiv P'(\sigma_k[\mathbf{o}])/\lambda_k[\mathbf{o}] \;.\end{aligned}$$

When $\mathcal{J}_k^+[\mathbf{o}]$ is empty, $\rho_k[\mathbf{o}]$ and $\nabla_k[\mathbf{o}]$ are set to ∞ and 0, respectively. $\nabla_k[\mathbf{o}]$ is called the *gradient* of $\mathcal{J}_k[\mathbf{o}]$.

```
procedure ENERGY_OPTIMAL_VOLTAGE_SCHEDULING($\mathcal{J}$, $\mathbf{o}$)
 1:  $\mathcal{J}' := \mathcal{J}$
 2:  $\mathbf{I} := [\min\{r_i | 1 \le i \le |\mathcal{J}|\}, \max\{d_i | 1 \le i \le |\mathcal{J}|\}]$   /* the whole execution interval */
 3:  $k := 1$          /* the iteration number */
 4:  while ( $\mathcal{J}' \neq \emptyset$ )
 5:      $\mathcal{T}_{\mathcal{J}'} := \{r_i, d_i | J_i \in \mathcal{J}'\}$
 6:      Find $r, d \in \mathcal{T}_{\mathcal{J}'}$ $(r < d)$ such that $\sigma_{[r,d]} = \frac{\sum_{J_i \in \mathcal{J}' \wedge [r_i, d_i] \subseteq [r,d]} o_i[\mathbf{o}]}{\|\mathbf{I} \cap [r,d]\|}$ is maximized.
          /* Ties are broken by preferring the largest $d - r$ and then the smallest $r$. */
 7:      $\mathcal{J}_k[\mathbf{o}] := \{J_i | [r_i, d_i] \subseteq [r,d]\}$   /* the set of jobs scheduled at the $k$-th iteration */
 8:      $\mathbf{I}_k[\mathbf{o}] := \mathbf{I} \cap [r,d]$
 9:      $\sigma_k[\mathbf{o}] := \sigma_{[r,d]}$       /* the constant speed allocated to jobs in $\mathcal{J}_k[\mathbf{o}]$ */
10:      foreach ($J_i \in \mathcal{J}_k[\mathbf{o}]$)
11:          $s_i[\mathbf{o}] := \sigma_k[\mathbf{o}]$        /* the constant speed allocated to $J_i$ */
12:          $a_i[\mathbf{o}] := o_i[\mathbf{o}] / s_i[\mathbf{o}]$   /* the execution time allocated to $J_i$ */
13:          $I_i[\mathbf{o}] :=$ the union of intervals in which $J_i$ is executed under the EDF policy
14:      end foreach
15:      $\mathcal{J}' := \mathcal{J}' - \mathcal{J}_k[\mathbf{o}]$
16:      $\mathbf{I} := \mathbf{I} - [r,d]$
17:      $k := k + 1$
18:  end while
19:  return $(a_1[\mathbf{o}], a_2[\mathbf{o}], \cdots, a_{|\mathcal{J}|}[\mathbf{o}])$   /* $\mathcal{S}[\mathbf{o}]$ can be directly computed from $\mathbf{A}[\mathbf{o}]$. */
end procedure
```

Fig. 2 Yao's algorithm to compute an energy-optimal voltage schedule

For a gradient g, $\mathcal{G}\langle g\rangle[\mathbf{o}]$ represents the subset of $\mathcal{G}[\mathbf{o}]$ that consists of job sets with gradient g, i.e.,

$$\mathcal{G}\langle g\rangle[\mathbf{o}] \overset{\text{def}}{=} \{\mathcal{J}_k[\mathbf{o}] \in \mathcal{G}[\mathbf{o}] \mid \nabla_k[\mathbf{o}] = g\} \ .$$

For workload tuples $\mathbf{o}_1$ and $\mathbf{o}_2$, we write $\mathbf{o}_1 \approx \mathbf{o}_2$ if $\mathcal{G}[\mathbf{o}_1] \equiv \mathcal{G}[\mathbf{o}_2]$.[4] For such $\mathbf{o}_1$ and $\mathbf{o}_2$, the shapes $\mathcal{S}[\mathbf{o}_1](t)$ and $\mathcal{S}[\mathbf{o}_2](t)$ are similar in that the speeds rise and sink at the same time-instants. Because $\mathbf{I}_{it_i[\mathbf{o}_1]}[\mathbf{o}_1] \equiv \mathbf{I}_{it_i[\mathbf{o}_2]}[\mathbf{o}_2]$ for any job J_i in $\mathcal{J}$, $\mathbf{A}[\mathbf{o}_2] = (a_1[\mathbf{o}_2], \ldots, a_{|\mathcal{J}|}[\mathbf{o}_2])$ can be expressed in terms of $\mathbf{o}_2$ and $\mathbf{A}[\mathbf{o}_1] = (a_1[\mathbf{o}_1], \ldots, a_{|\mathcal{J}|}[\mathbf{o}_1])$ as follows.

$$a_i[\mathbf{o}_2] = o_i[\mathbf{o}_2] \cdot \frac{\sum_{J_j \in \mathcal{J}_k[\mathbf{o}_1]} a_j[\mathbf{o}_1]}{\sum_{J_j \in \mathcal{J}_k[\mathbf{o}_1]} o_j[\mathbf{o}_2]} \quad \left(= o_i[\mathbf{o}_2] \cdot \frac{1}{s_i[\mathbf{o}_2]} \right) \quad \text{where } k = it_i[\mathbf{o}_1] \ . \tag{1}$$

[4] $\mathcal{G}[\mathbf{o}_1] \equiv \mathcal{G}[\mathbf{o}_2]$ does not always imply $\mathcal{J}_k[\mathbf{o}_1] \equiv \mathcal{J}_k[\mathbf{o}_2]$ for all $k \geq 1$. However, there exists a permutation π such that $\mathcal{J}_k[\mathbf{o}_1] \equiv \mathcal{J}_{\pi(k)}[\mathbf{o}_2]$ for all $k \geq 1$.

The relation $\approx$ is an equivalence relation and forms an (infinite) partition of $\mathcal{O}$. Generally, $\mathbf{A}[\mathbf{o}]$ is not explicitly represented in terms of $\mathbf{o}$. However, if $\mathbf{A}[\mathbf{o}']$ is available for some $\mathbf{o}'$ such that $\mathbf{o}' \approx \mathbf{o}$, the analytic expression of $\mathbf{A}[\mathbf{o}]$ can be obtained from Eq. (1). Our algorithm in Section 3 exploits this property in searching the optimal solution.

3 Optimal off-line algorithm

In this section, we present an optimal off-line algorithm for the problem. The algorithm starts by computing the energy-optimal voltage schedule $\mathbf{A}[\mathbf{u}]$ for the workload tuple $\mathbf{u}$. If $E(\mathbf{A}[\mathbf{u}]) \leq E_{\text{budget}}$, the algorithm returns $\mathbf{u}$ as the optimal solution since $\mathbf{A}[\mathbf{u}]$ satisfies the energy constraint (as well as timing constraints) and $F(\mathbf{u})$ is the upper bound of the total reward. Otherwise, the algorithm sets $\mathbf{o}$ to $\mathbf{u}$ and decreases $\mathbf{o}$ iteratively (but not beyond $\mathbf{m}$) until $E(\mathbf{A}[\mathbf{o}])$ reaches E_{budget}, as with the general descent method used for numerical optimization problems [3]. Figure 3 shows a snapshot of the algorithm progress. The challenges are how to determine the descent direction which varies continuously during search and how to make the search complete in polynomial time while guaranteeing the optimality.

A natural choice for the descent direction is the one that minimizes the decrease in $F(\mathbf{o})$ per unit decrease in $E(\mathbf{A}[\mathbf{o}])$ (equivalently, the one that maximizes the decrease in $E(\mathbf{A}[\mathbf{o}])$ per unit decrease in $F(\mathbf{o})$). However, the difficulty lies in the fact that $\mathbf{A}[\mathbf{o}]$ is not usually expressed explicitly in terms of $\mathbf{o}$, thus making it difficult to compute the differential of $E(\mathbf{A}[\mathbf{o}])$ in closed form. Furthermore, it is not obvious that the *greedy* gradient-based search always converges to the global optimal solution in our problem. The complicated solution space implicitly described by a condition in which $\mathbf{A}[\mathbf{o}]$ is involved also makes it difficult to determine the step size that yields a polynomial bound on the running time while still keeping the optimality.

To tackle the difficulties, we use the properties of energy-optimal voltage schedules described in Section 2.1. The gradient defined for a job J_i and a job set $\mathcal{J}_k[\mathbf{o}]$ corresponds to the energy decrease per unit reward decrease, and plays an important role in our algorithm. The procedure MAXIMIZE_REWARD in Fig. 4 describes the overall processing steps of our algorithm. The search is guided by the *global gradient* g. Initially, g is initially set to the largest gradient among those of jobs in $\cup_{k\geq 1}\mathcal{J}_k^+[\mathbf{u}]$. At each iteration, g is decreased to the level determined by the NEXT_SEPARATING_GRADIENT procedure.

The corresponding workload tuple $\mathbf{o}\langle g\rangle$ is initially set to $\mathbf{u}$ and is iteratively adjusted to the lower level by the procedure DECREASE_WORKLOADS (Fig. 5) such that each $g_i[\mathbf{o}\langle g\rangle]$ does not exceed the decreased g. (Note that $g_i[\mathbf{o}]$ decreases with $\mathbf{o}$.) After each invocation of the procedure DECREASE_WORKLOADS, the following invariant on $\mathbf{o}\langle g\rangle$ is preserved, which concisely describes the behavior of our algorithm:

$$o_i[\mathbf{o}\langle g-\Delta g\rangle] = \begin{cases} o_i[\mathbf{o}\langle g\rangle] & g_i[\mathbf{o}\langle g\rangle] < g \vee o_i[\mathbf{o}\langle g\rangle] = m_i, \\ o_i[\mathbf{o}\langle g\rangle] - \Delta o_i & g_i[\mathbf{o}\langle g\rangle] = g \wedge o_i[\mathbf{o}\langle g\rangle] > m_i. \end{cases}$$

where Δg is a sufficiently small value and Δo_i's (> 0) satisfy

$$g_i[\mathbf{o}\langle g-\Delta g\rangle] = g - \Delta g \quad \text{for all } i \text{ s.t. } g_i[\mathbf{o}\langle g\rangle] \geq g \ \wedge \ o_i[\mathbf{o}\langle g\rangle] > m_i. \tag{2}$$

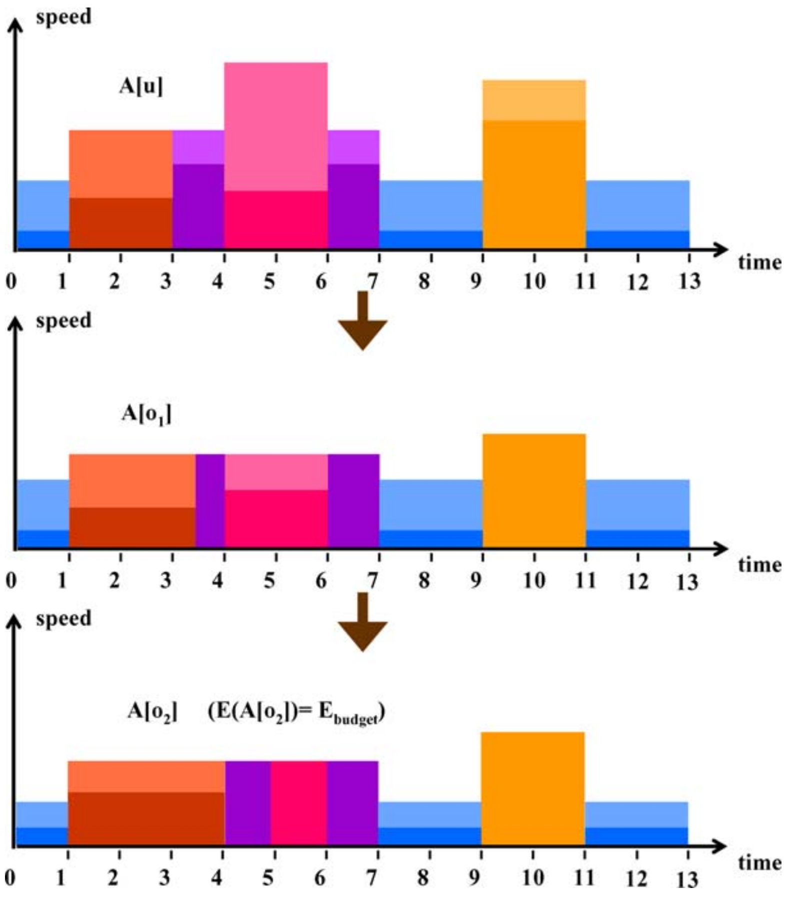

Fig. 3 A step-by-step snapshot of the algorithm progress

Note that $g_i[\mathbf{o}\langle g-\Delta g\rangle]$ is not generally given as an analytic expression which is necessary in solving Eq. (2). Informally, $\Delta\mathbf{o} = (\Delta o_1, \Delta o_2, \ldots, \Delta o_{|\mathcal{J}|})$ (Δo_i is set to 0 for all i such that $g_i[\mathbf{o}\langle g\rangle] < g$ or $o_i[\mathbf{o}\langle g\rangle] = m_i$.) satisfying Eq. (2) represents the search direction at $\mathbf{o}\langle g\rangle$ that results in the biggest decrease in the energy per unit decrease in the total reward.

Let us assume that $\mathbf{o}\langle g-\Delta g\rangle \approx \mathbf{o}\langle g\rangle$, i.e., $\mathcal{G}[\mathbf{o}\langle g-\Delta g\rangle] \equiv \mathcal{G}[\mathbf{o}\langle g\rangle]$. Then, we can obtain an analytic expression for $g_i[\mathbf{o}]$ in terms of $\mathbf{A}[\mathbf{o}\langle g\rangle]$ and $\Delta\mathbf{o} = \{\Delta o_1, \ldots, \Delta o_{|\mathcal{J}|}\}$. From Eq. (1), $s_i[\mathbf{o}\langle g-\Delta g\rangle]$ is given by

$$s_i[\mathbf{o}\langle g-\Delta g\rangle] = \frac{\sum_{J_j\in\mathcal{J}'}(o_j[\mathbf{o}\langle g\rangle]-\Delta o_j)}{\sum_{J_j\in\mathcal{J}'} a_j[\mathbf{o}\langle g\rangle]} \quad \text{where } \mathcal{J}' = \mathcal{J}_{it_i[\mathbf{o}\langle g\rangle]}[\mathbf{o}\langle g\rangle],$$

and $g_i[\mathbf{o}\langle g-\Delta g\rangle]$ is given in terms of $s_i[\mathbf{o}\langle g-\Delta g\rangle]$ by

$$g_i[\mathbf{o}\langle g-\Delta g\rangle] = \frac{P'(s_i[\mathbf{o}\langle g-\Delta g\rangle])}{\rho_i'(o_i[\mathbf{o}\langle g\rangle]-\Delta o_i)}, \tag{3}$$

```
procedure MAXIMIZE_REWARD
 1:  $E := E(\mathcal{S}[\mathbf{u}])$
 2:  $g := \max\{g_i[\mathbf{u}] | J_i \in \cup_{k \geq 1} \mathcal{J}_k^+[\mathbf{u}]\}$
 3:  $\mathbf{o}\langle g \rangle := \mathbf{u}$
 4:  while $(E > E_{\text{budget}})$
 5:      $\mathbf{o} := \mathbf{o}\langle g \rangle$
 6:      $g_s := g$
 7:      $g :=$ NEXT_SEPARATING_GRADIENT$(\mathbf{o}\langle g \rangle, g)$
 8:      $\mathbf{o}\langle g \rangle :=$ DECREASE_WORKLOADS$(\mathbf{o}\langle g \rangle, g)$
 9:      $E := E(\mathbf{A}[\mathbf{o}\langle g \rangle])$
10:  end while
     /* The optimal solution is between $\mathbf{o}\langle g \rangle$ and $\mathbf{o}$. */
11:  Solve the simultaneous equations given by Eq.(5) for all $\mathcal{J}_k[\mathbf{o}] \in \mathcal{G}\langle g_s \rangle[\mathbf{o}]$ and
     $\sum_{\mathcal{J}_k[\mathbf{o}] \in \mathcal{G}\langle g_s \rangle[\mathbf{o}]} P\left(\frac{\sum_{J_i \in \mathcal{J}_k[\mathbf{o}]} \gamma_i(h_k)}{\|\mathbf{I}_k\|}\right) \cdot \|\mathbf{I}_k\| + \sum_{\mathcal{J}_k[\mathbf{o}] \in \mathcal{G}[\mathbf{o}] - \mathcal{G}\langle g_s \rangle[\mathbf{o}]} P\left(\frac{\sum_{J_i \in \mathcal{J}_k[\mathbf{o}]} o_i[\mathbf{o}]}{\|\mathbf{I}_k\|}\right) \cdot \|\mathbf{I}_k\| = E_{\text{budget}}.$
12:  foreach $(J_i \in \mathcal{J})$
13:      if $(\mathcal{J}_{it_i[\mathbf{o}]} \in \mathcal{G}\langle g_s \rangle[\mathbf{o}])$
14:          $o_i := \gamma_i(h_{it_i[\mathbf{o}]})$
15:      else
16:          $o_i := o_i[\mathbf{o}]$
17:      end if
18:  end foreach
19:  return $(o_1, o_2, \cdots, o_{|\mathcal{J}|})$   /* $E(\mathcal{S}[(o_1, o_2, \cdots, o_{|\mathcal{J}|})]) = E_{\max}$ and $\sum_{i=1}^{|\mathcal{J}|} f(o_i)$ is maximum. */
end procedure
```

Fig. 4 The optimal off-line reward-based voltage scheduling algorithm

```
procedure DECREASE_WORKLOADS( $\mathbf{o}$, $g$)
     /* The procedure NEXT_SEPARATING_GRADIENT guarantees $\mathbf{o} \approx \mathbf{o}\langle g \rangle$. */
 1:  $(o_1, o_2, \cdots, o_{|\mathcal{J}|}) := \mathbf{o}$
 2:  foreach $(\mathcal{J}_k[\mathbf{o}] \in \mathcal{G}\langle g \rangle[\mathbf{o}])$
 3:      Solve the equation given by Eq.(5).
 4:      foreach $(J_i \in \mathcal{J}_k[\mathbf{o}])$
 5:          $o_i := \gamma_i(h_k)$
 6:      end foreach
 7:  end foreach
 8:  return $(o_1, o_2, \cdots, o_{|\mathcal{J}|})$
end procedure
```

Fig. 5 The algorithm to decrease the workload tuple for a given global gradient

which is an explicit expression of $\Delta\mathbf{o}$. Therefore, we can compute $\Delta\mathbf{o}$ by solving Eq. (2) either numerically or analytically. By repeating this process, we can obtain $\mathbf{o}\langle g\rangle$ for all $0 < g < \nabla[\mathbf{u}]$, however, it requires infinitely many steps because Δg is assumed to be arbitrarily small.

To bring the number of steps down to a polynomial, we exploit the fact that an equivalence class under the relation $\approx$ covers sufficiently large range of $\mathbf{o}$, i.e., $\{\mathbf{o}\langle g\rangle \mid 0 < g \leq \nabla[\mathbf{u}]\}$ is partitioned into a polynomial number of ranges $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_n$:

$$\{\mathbf{o}\langle g\rangle \mid 0 < g \leq \nabla[\mathbf{u}]\} = \cup_{l=1}^{n}\mathcal{O}_l = \cup_{l=1}^{n}\{\mathbf{o}\langle g\rangle \mid g_{l-1} < g \leq g_l\}.$$

where $g_0 = 0$ and $g_n = \nabla[\mathbf{u}]$. We call $g_1, g_2, \ldots, g_{n-1}$ *separating* gradients. For the time being, assume that the number of separating gradients is bounded by a polynomial and that each one can be found in polynomial time. (We will prove these assumptions later in this section.) Then, if $\mathbf{o}\langle g\rangle$ can be found for all $g_{l-1} < g \leq g_l$ in polynomial time, we can obtain $\mathbf{o}\langle g\rangle$ for all $0 < g < \nabla[\mathbf{u}]$ in polynomial time.

For a given $\mathbf{o}\langle g_l\rangle$, we describe how to compute $\mathbf{o}\langle g\rangle$ for $g_{l-1} < g \leq g_l$. From the definition of g_{l-1} and g_l, we have $\mathbf{o}\langle g_l\rangle \approx \mathbf{o}\langle g\rangle$, i.e., $\mathcal{G}[\mathbf{o}\langle g_l\rangle] \equiv \mathcal{G}[\mathbf{o}\langle g\rangle]$. Without loss of generality, we assume that $\mathcal{J}_k[\mathbf{o}\langle g_l\rangle] \equiv \mathcal{J}_k[\mathbf{o}\langle g\rangle]$ for all $k \geq 1$. Then, we have

$$\mathbf{I}_k[\mathbf{o}\langle g_l\rangle] \equiv \mathbf{I}_k[\mathbf{o}\langle g\rangle] \quad \text{and} \quad \sum\nolimits_{J_i \in \mathcal{J}_k[\mathbf{o}\langle g_l\rangle]} a_i[\mathbf{o}\langle g_l\rangle] = \sum\nolimits_{J_i \in \mathcal{J}_k[\mathbf{o}\langle g\rangle]} a_i[\mathbf{o}\langle g\rangle].$$

For brevity, $\mathcal{J}_k$, $\mathbf{I}_k$ and $\|\mathbf{I}_k\|$ are used to denote $\mathcal{J}_k[\mathbf{o}\langle g_l\rangle]$, $\mathbf{I}_k[\mathbf{o}\langle g_l\rangle]$ and $\|\mathbf{I}_k[\mathbf{o}\langle g_l\rangle]\|$, respectively. For each $J_i \in \mathcal{J}_k$, its workload $o_i[\mathbf{o}\langle g\rangle]$ is set to an explicit expression of h_k (h_k represents the decrease in the reward per unit decrease in the workload):

$$o_i[\mathbf{o}\langle g\rangle] = \gamma_i(h_k) \quad \text{where } \gamma_i(\cdot) \text{ is the function defined by} \tag{4}$$

$$\gamma_i(x) = \begin{cases} u_i & 0 < x < \rho_i'(u_i), \\ \rho_i'^{-1}(x) & \rho_i'(u_i) \leq x \leq \rho_i'(m_i), \\ m_i & x > \rho_i'(m_i). \end{cases}$$

h_k satisfies the following equation that corresponds to Eq. (2):

$$\frac{P'\left(\sum_{J_i \in \mathcal{J}_k} \gamma_i(h_k)/\|\mathbf{I}_k\|\right)}{h_k} = g. \tag{5}$$

Because the left-hand side of Eq. (5) is a strictly decreasing function of h_k (from the concavity of ρ_i), h_k is uniquely determined, and so are $o_i[\mathbf{o}\langle g\rangle]$'s. The procedure DECREASE_WORKLOADS takes as inputs $\mathbf{o}\langle g_l\rangle$ and g such that $\mathbf{o}\langle g_l\rangle \approx \mathbf{o}\langle g\rangle$ and returns $\mathbf{o}\langle g\rangle$. We later prove that $\mathbf{o}\langle g\rangle$ always passes through the optimal solution, i.e., the optimal solution $\mathbf{o}_{\text{opt}}$ is given by $\mathbf{o}\langle g_{\text{H.-S.Yun and J.Kimopt}}\rangle$ where g_{opt} is the unique gradient satisfying $E(\mathbf{A}[\mathbf{o}\langle g_{\text{opt}}\rangle]) = E_{\text{budget}}$.

To show that our algorithm runs in polynomial time, we now describe how to compute each separating gradient in polynomial time and prove that the number of separating points is bounded by a polynomial. Let us consider necessary conditions for a global gradient g to a separating gradient. Suppose that g is a separating gradient. Then,

(a) jobs in $\mathcal{J}_{k_1}[\mathbf{o}\langle g+\varepsilon\rangle]$ and $\mathcal{J}_{k_2}[\mathbf{o}\langle g+\varepsilon\rangle]$ are merged into $\mathcal{J}_k[\mathbf{o}\langle g-\varepsilon\rangle]$, or
(b) jobs in $\mathcal{J}_k[\mathbf{o}\langle g+\varepsilon\rangle]$ is divided into $\mathcal{J}_{k_1}[\mathbf{o}\langle g-\varepsilon\rangle]$ and $\mathcal{J}_{k_2}[\mathbf{o}\langle g-\varepsilon\rangle]$

where ε is the infinitesimal. Both cases may occur simultaneously.

The necessary condition for the case (a) to occur is given by

$$\begin{aligned}
&\lim_{\varepsilon\to 0}\sigma_{k_1}[\mathbf{o}\langle g+\varepsilon\rangle] = \lim_{\varepsilon\to 0}\sigma_{k_2}[\mathbf{o}\langle g+\varepsilon\rangle], \quad \text{which implies}\\
&\lim_{\varepsilon\to 0}\frac{\sum_{J_i\in\mathcal{J}_{k_1}[\mathbf{o}\langle g+\varepsilon\rangle]} o_i[\mathbf{o}\langle g+\varepsilon\rangle]}{\sum_{J_i\in\mathcal{J}_{k_1}[\mathbf{o}\langle g+\varepsilon\rangle]} a_i[\mathbf{o}\langle g+\varepsilon\rangle]} = \lim_{\varepsilon\to 0}\frac{\sum_{J_i\in\mathcal{J}_{k_2}[\mathbf{o}\langle g+\varepsilon\rangle]} o_i[\mathbf{o}\langle g+\varepsilon\rangle]}{\sum_{J_i\in\mathcal{J}_{k_2}[\mathbf{o}\langle g+\varepsilon\rangle]} a_i[\mathbf{o}\langle g+\varepsilon\rangle]}. \qquad (6)
\end{aligned}$$

The second case is more complicated. For a job $J_i \in \mathcal{J}_k[\mathbf{o}\langle g+\varepsilon\rangle]$, let I_i' denote $\mathbf{I}_k[\mathbf{o}\langle g+\varepsilon\rangle] \cap [r_i, d_i]$ and let $\mathcal{J}_{[r,d]}$ denote $\{J_i \in \mathcal{J}_k[\mathbf{o}\langle g+\varepsilon\rangle] \mid I_i' \subseteq [r,d]\}$. Then, the necessary condition for the case (b) is given by

$$\begin{aligned}
&\lim_{\varepsilon\to 0}\sigma_{k_1}[\mathbf{o}\langle g-\varepsilon\rangle] = \lim_{\varepsilon\to 0}\sigma_{k_2}[\mathbf{o}\langle g-\varepsilon\rangle], \quad \text{which implies}\\
&\exists\, r,\ d \in \{\min I_i',\ \max I_i' \mid J_i \in \mathcal{J}_k[\mathbf{o}\langle g+\varepsilon\rangle]\ \},\\
&\lim_{\varepsilon\to 0}\frac{\sum_{J_i\in\mathcal{J}_{[r,d]}} o_i[\mathbf{o}\langle g+\varepsilon\rangle]}{\|\mathbf{I}_k[\mathbf{o}\langle g+\varepsilon\rangle]\cap[r,d]\|} = \lim_{\varepsilon\to 0}\frac{\sum_{J_i\in\mathcal{J}_k[\mathbf{o}\langle g+\varepsilon\rangle]-\mathcal{J}_{[r,d]}} o_i[\mathbf{o}\langle g+\varepsilon\rangle]}{\|\mathbf{I}_k[\mathbf{o}\langle g+\varepsilon\rangle]\| - \|\mathbf{I}_k[\mathbf{o}\langle g+\varepsilon\rangle]\cap[r,d]\|}. \qquad (7)
\end{aligned}$$

Provided that the separating gradients $g_n, g_{n-1}, \ldots, g_l$ are identified, the next lower separating gradient g_{l-1} can be found by the following procedure:

(a) Replace $\sum_{J_i\in\mathcal{J}_{k_1}[\mathbf{o}\langle g+\varepsilon\rangle]} a_i[\mathbf{o}\langle g+\varepsilon\rangle]$ and $\sum_{J_i\in\mathcal{J}_{k_2}[\mathbf{o}\langle g+\varepsilon\rangle]} a_i[\mathbf{o}\langle g+\varepsilon\rangle]$ in Eq. (6) by $\sum_{J_i\in\mathcal{J}_{k_1}[\mathbf{o}\langle g_l\rangle]} a_i[\mathbf{o}\langle g_l\rangle]$ and $\sum_{J_i\in\mathcal{J}_{k_2}[\mathbf{o}\langle g_l\rangle]} a_i[\mathbf{o}\langle g_l\rangle]$, respectively. (Note that the latter two are known values, since g_l is already known.)
(b) Replace $\mathbf{I}_k[\mathbf{o}\langle g+\varepsilon\rangle]$ in Eq. (7) by $\mathbf{I}_k[\mathbf{o}\langle g_l\rangle]$ (Note that the latter is already known.)
(c) Remove lim operators from Eqs. (6) and (7) and replace $g+\varepsilon$ by g.
(d) Return the largest g ($< g_l$) that satisfies the simultaneous Eqs. (4)–(6), or the Eqs. (4), Eq. (5) and Eq. (7).

The above procedure makes good use of the property that $\mathbf{o}\langle g\rangle \approx \mathbf{o}\langle g_l\rangle$ for all $g_{l-1} < g \le g_l$ and $\mathbf{o}\langle g_l\rangle \equiv \lim_{\varepsilon\to 0}\mathbf{o}\langle g_l+\varepsilon\rangle$. At each iteration, the procedure NEXT_SEPARATING_GRADIENT computes the next lower separating gradient in this way. It remains to show that the number of separating gradients is bounded by a polynomial. In proving this property, we exploit the fact that the order on speed levels of jobs is not changed too frequently. (Refer to [21] for the proof.)

Lemma 3.1. *The number of separating gradients within* $(0, \nabla[\mathbf{u}]]$ *is bounded by* $4\cdot|\mathcal{J}|^2$.

We now prove that our algorithm always computes a maximum-reward schedule for a given energy budget. The proof is based on the rationale that the *gradient* of each job is as uniform as possible under an optimal solution. In the following, we formalize the notion of uniform gradient and establish the link between the uniform gradient and maximum-reward schedules.

Definition 3.2. For a schedule $\langle \mathbf{A}, \mathbf{o}\rangle = \langle (a_1, \ldots, a_{|\mathcal{J}|}), (o_1, \ldots, o_{|\mathcal{J}|})\rangle \in \mathcal{F}_{\mathcal{J}} \times \mathcal{O}_{\mathcal{J}}$ and $g > 0$, we say that $\langle \mathbf{A}, \mathbf{o}\rangle$ *has a uniform gradient* g, written $\langle \mathbf{A}, \mathbf{o}\rangle \underline{\underline{\nabla}}\, g$, if

$$\begin{aligned}
&\text{(a)}\quad \forall J_i \in \mathcal{J} \text{ s.t. } o_i = u_i,\quad g_i(a_i, o_i) \le g,\\
&\text{(b)}\quad \forall J_i \in \mathcal{J} \text{ s.t. } o_i = m_i,\quad g_i(a_i, o_i) \ge g \quad \text{and}\\
&\text{(c)}\quad \forall J_i \in \mathcal{J} \text{ s.t. } m_i < o_i < u_i,\quad g_i(a_i, o_i) \equiv g
\end{aligned}$$

where $g_i(a_i, o_i) \overset{\text{def}}{=} P'(o_i/a_i)/\rho_i'(o_i)$.

Assume that the job J_5 in Fig. 1 has $u_5 = 5$. Then, we can easily check that the schedule in Fig. 1(b) has a uniform gradient $3 \cdot (51/4)^{2/3}$. However, the schedule in Fig. 1(a) does not have a uniform gradient (i.e., $g_1(a_1, o_1) = 12 \ne g_2(a_2, o_2) = 27$). Note that both schedules consume the same amount of energy but the total reward of the schedule with the uniform gradient in Fig. 1(b) is larger. Intuitively, uniform gradient is to maximizing the total reward what flat speed is to minimizing the total energy consumption.

Let $\mathcal{H}$ denote the set of maximum-reward schedules for varying values of the energy budget e:

$$\begin{aligned}
\mathcal{H} \overset{\text{def}}{=} \Big\{&\langle \mathbf{A}, \mathbf{o}\rangle \in \mathcal{F}_{\mathcal{J}} \times \mathcal{O}_{\mathcal{J}} \;\Big|\; \exists\, E(\langle \mathbf{A}[\mathbf{m}], \mathbf{m}\rangle) \le e \le E(\langle \mathbf{A}[\mathbf{u}], \mathbf{u}\rangle),\\
&\langle \mathbf{A}, \mathbf{o}\rangle \text{ is the maximum-reward schedule for } e.\}\\
\equiv \Big\{&\langle \mathbf{A}, \mathbf{o}\rangle \in \mathcal{F}_{\mathcal{J}} \times \mathcal{O}_{\mathcal{J}} \;\Big|\; \forall \langle \mathbf{A}', \mathbf{o}'\rangle \in \mathcal{F}_{\mathcal{J}} \times \mathcal{O}_{\mathcal{J}} \text{ s.t. } E(\langle \mathbf{A}', \mathbf{o}'\rangle) \le E(\langle \mathbf{A}, \mathbf{o}\rangle),\\
&F(\mathbf{o}') \le F(\mathbf{o})\Big\}.
\end{aligned}$$

We will prove that $\langle \mathbf{A}, \mathbf{o}\rangle \in \mathcal{H}$ if and only if $\mathbf{A} \equiv \mathbf{A}[\mathbf{o}]$ and $\langle \mathbf{A}, \mathbf{o}\rangle$ has a global gradient, and then develop an algorithm that takes $g > 0$ as an input and computes the schedule with the uniform gradient g. To begin with, we consider the necessity part of the condition for $\langle \mathbf{A}, \mathbf{o}\rangle \in \mathcal{H}$. (Refer to [21] for the proof.)

Lemma 3.3. *For any schedule* $\langle \mathbf{A}, \mathbf{o}\rangle = \langle (a_1, \ldots, a_{|\mathcal{J}|}), (o_1, \ldots, o_{|\mathcal{J}|})\rangle \in \mathcal{H}$, $\mathbf{A} \equiv \mathbf{A}[\mathbf{o}]$ *and* $\langle \mathbf{A}, \mathbf{o}\rangle$ *has a uniform gradient.*

For a schedule $\langle \mathbf{A}, \mathbf{o}\rangle$ such that $\langle \mathbf{A}, \mathbf{o}\rangle \underline{\underline{\nabla}}\, g$, we can derive the following relationship between $\mathbf{A}$ and $\mathbf{o}$ from Definition 3.2. (Refer to [21] for the proof.)

Lemma 3.4. *Let* $\langle \mathbf{A}, \mathbf{o}\rangle = \langle (a_1, \ldots, a_{|\mathcal{J}|}), (o_1, \ldots, o_{|\mathcal{J}|})\rangle \in \mathcal{F}_{\mathcal{J}} \times \mathcal{O}_{\mathcal{J}}$ *be a schedule such that* $\langle \mathbf{A}, \mathbf{o}\rangle \underline{\underline{\nabla}}\, g$. *Then,* $a_i \equiv \mu_i\langle g\rangle(o_i/a_i)$ *for all* $1 \le i \le |\mathcal{J}|$ *where*

$$\mu_i\langle g\rangle(x) \overset{\text{def}}{=} \begin{cases} u_i/x & x \le (P')^{-1}(g \cdot \rho_i'(u_i)),\\ (\rho_i')^{-1}(P'(x)/g)/x & (P')^{-1}(g \cdot \rho_i'(u_i)) < x < (P')^{-1}(g \cdot \rho_i'(m_i)),\\ m_i/x & x \ge (P')^{-1}(g \cdot \rho_i'(m_i)). \end{cases}$$

We are ready to show that the converse of Lemma 3.3 also holds, implying that $\langle \mathbf{A}[\mathbf{o}], \mathbf{o}\rangle$ is a maximum-reward schedule if and only if $\langle \mathbf{A}[\mathbf{o}], \mathbf{o}\rangle$ has a uniform gradient. (Refer to [21] for the proof.)

Lemma 3.5. *Given $g > 0$, there is a unique $\mathbf{o} \in \mathcal{O}_{\mathcal{J}}$ such that $\langle \mathbf{A}[\mathbf{o}], \mathbf{o} \rangle \underline{\nabla} g$.*

Lemma 3.6. *For any schedule $\langle \mathbf{A}[\mathbf{o}], \mathbf{o} \rangle \in \mathcal{F}_{\mathcal{J}} \times \mathcal{O}_{\mathcal{J}}$ with a uniform gradient, $\langle \mathbf{A}[\mathbf{o}], \mathbf{o} \rangle \in \mathcal{H}$.*

Proof: Consider a schedule $\langle \mathbf{A}[\mathbf{o}], \mathbf{o} \rangle$ such that $\langle \mathbf{A}[\mathbf{o}], \mathbf{o} \rangle \underline{\nabla} g$ for some $g > 0$. Suppose to the contrary that $\langle \mathbf{A}[\mathbf{o}], \mathbf{o} \rangle \notin \mathcal{H}$. Then, there must exist $\mathbf{o}' \in \mathcal{O}_{\mathcal{J}}$ such that $\mathbf{o}' \neq \mathbf{o}$ and $\langle \mathbf{A}[\mathbf{o}'], \mathbf{o}' \rangle \underline{\nabla} g$, which contradicts Lemma 3.5. Therefore, $\langle \mathbf{A}, \mathbf{o} \rangle \in \mathcal{H}$. □

Finally, we prove the optimality of the algorithm in Section 3.

Theorem 3.7. *The procedure* MAXIMIZE_REWARD *in Fig. 4 always returns an optimal solution.*

Proof: Consider $\mathbf{o}\langle g \rangle$ computed by the algorithm in Section 3. We would like to show that $\langle \mathbf{A}[\mathbf{o}\langle g \rangle], \mathbf{o}\langle g \rangle \rangle \underline{\nabla} g$.

For $J_i \in \mathcal{J}$ such that $o_i[\mathbf{o}\langle g \rangle] = u_i$, it must be the case that $\rho_i'(u_i) \geq h_k$. Then,

$$g_i(a_i[\mathbf{o}\langle g \rangle], o_i[\mathbf{o}\langle g \rangle]) = P'(s_i[\mathbf{o}\langle g \rangle])/\rho_i'(u_i) \leq P'(s_i[\mathbf{o}\langle g \rangle])/h_k = g.$$

Similarly, $g_i(a_i[\mathbf{o}\langle g \rangle], o_i[\mathbf{o}\langle g \rangle]) \geq g$ for $J_i \in \mathcal{J}$ such that $o_i[\mathbf{o}\langle g \rangle] = m_i$. Finally, for $J_i \in \mathcal{J}$ such that $m_i < o_i[\mathbf{o}\langle g \rangle] < u_i$, it must be the case that $\rho_i'(u_i) < h_k < \rho_i'(m_i)$. Thus,

$$\begin{aligned} g_i(a_i[\mathbf{o}\langle g \rangle], o_i[\mathbf{o}\langle g \rangle]) &= P'(s_i[\mathbf{o}\langle g \rangle])/\rho_i'(\gamma_i(h_k)) = P'(s_i[\mathbf{o}\langle g \rangle])/\rho_i'(\rho_i'^{-1}(h_k)) \\ &= P'(s_i[\mathbf{o}\langle g \rangle])/h_k = g, \end{aligned}$$

and we finally have $\langle \mathbf{A}[\mathbf{o}\langle g \rangle], \mathbf{o}\langle g \rangle \rangle \underline{\nabla} g$. Consequently, from Lemma 3.6, $\langle \mathbf{A}[\mathbf{o}\langle g \rangle], \mathbf{o}\langle g \rangle \rangle \in \mathcal{H}$, and $\langle \mathbf{A}[\mathbf{o}\langle g \rangle], \mathbf{o}\langle g \rangle \rangle$ is the maximum-reward schedule for the given energy budget E_{budget}.

□

4 On-Line algorithm

The off-line algorithm described in Section 3 is based on the assumption that the exact workloads (i.e., the mandatory workload and the upper bound of the optional workload) are known in advance. Thus, the off-line algorithm can be applied to the case when all the jobs finish at their worst-case execution cycles. Furthermore, it can be used to compute the theoretical lower bound with the complete execution trace information (i.e., the actual workloads), which is useful in evaluating the performance of on-line scheduling algorithm. However, the workload of each job varies, sometimes by a large amount, which cannot be adequately handled by off-line scheduling alone. On-line scheduling is effective in leveraging workload variations, and we consider an on-line algorithm for the reward-based voltage scheduling problem.

On-line reward-based voltage scheduling differs from conventional on-line voltage scheduling in that the energy consumption is not given as an optimization goal, but as a constraint. Furthermore, the optimization goal is to maximize the sum of rewards associated with optional workloads. Therefore, our on-line algorithm manages *energy slack* as well as time slack. Informally, the energy slack is the residual energy reserved by an unexpected lower speed or idle time. For example, assume that the energy required by an off-line schedule within the interval $[0, t]$ is given by $E(t)$ and the energy actually used at runtime is given by $E'(t)$ where $E'(t) \leq E(t)$. Then, the amount of energy slack reserved at time t is defined to be $E(t) - E'(t)$. The energy slack is much easier to manage than time slack because it can be directly detected and distributed among jobs executing next while for the time slack the preemption driven by the priority makes the analysis complicated.

With regard to time slack management, conventional voltage scheduling consists of two parts: slack estimation and slack distribution. The goal of the slack estimation part is to identify as much available time as possible while the goal of the slack distribution part is to distribute the time slack so that the resulting voltage schedule is as flat as possible. For the time slack estimation, we adopt the existing method for fixed-priority tasks developed by Gruian [6], which is based on the priority-based slack stealing method [10]. However, in distributing the slack, we consider both the energy slack and the time slack, and try to increase the reward as much as possible by fully utilizing the energy slack as well as time slack.

In distributing two kinds of slacks, we exploit two properties that an optimal off-line schedule exhibits (see Lemma 3.3.). First, the voltage schedule (as a function of time) should be as flat as possible; a maximum-reward schedule for a given energy budget is also a minimum-energy schedule among those with the same workload tuple. Second, the gradients of jobs (i.e., $P'(s_i)/\rho_i'(o_i)$) should also be as uniform as possible.

Assume that the time slack Δt and the energy slack ΔE are available at t and can be distributed among jobs $J_{i_1}, J_{i_2}, \ldots, J_{i_n}$ in the ready queue. Let a_{i_j} and s_{i_j} be the allowed execution time and the speed, respectively, determined by the off-line scheduler. Then, the on-line scheduler tries to obtain an approximate solution for the following problem:

Find Δa_{i_j} and Δs_{i_j} for $j = 1, 2, \ldots, n$ such that

$$\frac{P'(s_{i_j} + \Delta s_{i_j})}{\rho'_{i_j}((s_{i_j} + \Delta s_{i_j}) \cdot (a_{i_j} + \Delta a_{i_j}))} \tag{8}$$

is as uniform as possible subject to

$$\Delta t \geq \sum_{j=1}^{n} \Delta a_{i_j} \quad \text{and} \quad \Delta E \geq \sum_{j=1}^{n} P(s_{i_j} + \Delta s_{i_j}) \cdot (a_{i_j} + \Delta a_{i_j}) - P(s_{i_j}) \cdot a_{i_j}. \tag{9}$$

From the convexity of P and the concavity of ρ, the gradient of J_{i_j} given by Eq. (8) increases both with Δa_{i_j} and with Δs_{i_j}. Thus, it is natural to assign larger Δa_{i_j} and Δs_{i_j} to a job with lower gradient. Our on-line algorithm first distributes Δt by incrementing Δa_{i_j}'s iteratively until $\sum_{j=1}^{n} \Delta a_{i_j}$ reaches the available slack time Δt and then increments Δs_{i_j}'s to distribute the remaining energy slack as in Fig. 7.

Figure 6 showan overall implementation of the on-line scheduler. The scheduler manages the following time-varying state variables associated with each job:

- $time\_left_i$: the remaining execution time of J_i.
- $workload_i$ and $workload\_left_i$: the total workload and the remaining workload of J_i.
- $speed_i$: the speed of J_i. The scheduler always updates $speed_i$ to $workload\_left_i /\ time\_left_i$.
- $energy\_left_i$: the remaining energy that can be used by J_i. The scheduler always updates $energy\_left_i$ to $time\_left_i \cdot P(speed_i)$.

```
procedure INITIALIZE
    $\langle(a_1,\cdots,a_{|\mathcal{J}|}),(o_1,\cdots,o_{|\mathcal{J}|})\rangle$ := an off-line job-level schedule    /* A task-level schedule can be substituted. */
    $energy\_slack := 0$
    $time\_slack_i := 0$  for $1 \le i \le |\mathcal{J}|$
end procedure

procedure UPON_RELEASE ($J_i$)
    $workload\_left_i := o_i$ , $workload_i := o_i$
    $time\_left_i := a_i$ , $speed_i := o_i/a_i$
    $energy\_left_i := a_i \cdot P(speed_i)$  /* the energy allocated to $J_i$ */
    /* If $J_i$ has a higher priority than the currently executing job and the jobs in the ready queue,
       the on-line scheduler will invoke the procedure UPON_DISPATCH to execute $J_i$. */
end procedure

procedure UPON_DISPATCH ($J_i$)
    INVALIDATE_TIME_SLACK ($J_i$)
    Run $J_i$ at the speed $speed_i$.    /* $speed_i$ is always updated to $workload\_left_i/time\_left_i$. */
end procedure

procedure DURING_EXECUTION ($J_i$,$\Delta t$)    /* $J_i$ has been executed for $\Delta t$ without any interruption. */
    $workload\_left_i$ -= $speed_i \cdot \Delta t$
    $time\_left_i$ -= $\Delta t$ , $energy\_left_i$ -= $\Delta t \cdot P(speed_i)$
end procedure

procedure UPON_COMPLETE ($J_i$)
    RECLAIM_SLACKS ($J_i$)        /* Reclaim the unused energy left by $J_i$. */
    if (there is no job in the ready queue)        /* Spend idle time until the next arrival time. */
        DURING_IDLE_TIME (the length of the idle interval)
    end if
    let $\mathcal{J}^{\text{ready}}$ be the set of jobs in the ready queue.
    DISTRIBUTE_SLACKS ($\mathcal{J}^{\text{ready}}$)        /* Distribute the slacks among jobs in the ready queue. */
    if (there is only one job $J_{i_r}$ in $\mathcal{J}^{\text{ready}}$)
        STRETCH_TO_NTA ($J_{i_r}$)        /* $J_{i_r}$ can safely use more time slack. */
    end if
    /* Upon return, the on-line scheduler will invoke the procedure UPON_DISPATCH to execute the highest priority job in the ready queue. */
end procedure

procedure INVALIDATE_TIME_SLACK ($J_i$)
    $time\_slack_j := 0$  for $1 \le j \le i$
end procedure

procedure DURING_IDLE_TIME ($\Delta t$)
    $time\_slack_j := \max\{0, time\_slack_j - \Delta t\}$  for $1 \le j \le |\mathcal{J}|$
end procedure

procedure RECLAIM_SLACKS ($J_i$)
    $energy\_slack$ += $energy\_left_i$
    $time\_slack_j$ += $time\_left_i$  for $i+1 \le j \le |\mathcal{J}|$
    $time\_left_i := 0$  /* This assignment is used only for the correctness proof; the on-line scheduler destroys the variable at this point. */
end procedure
```

Fig. 6 The on-line scheduling algorithm

The on-line scheduler starts with a feasible off-line schedule $\langle \mathbf{A}, \mathbf{o}\rangle = \langle(a_1, \ldots, a_{|\mathcal{J}|}), (o_1, \ldots, o_{|\mathcal{J}|})\rangle$. (Note that a task-level schedule (e.g., as in [16]) can be substituted for the job-level schedule $\langle \mathbf{A}, \mathbf{o}\rangle$ in our on-line algorithm.) When a job J_i is released, the associated state variables are initialized according to the off-line schedule by the UPON_RELEASE procedure. During the execution of J_i, the variable *time_left*$_i$ is decreased at the same rate as time passes and *workload_left*$_i$ and *energy_left*$_i$ are updated accordingly (as in the procedure DURING_EXECUTION). When J_i finishes its execution, the procedure RECLAIM_SLACKS is invoked to collect the unused time (i.e., *time_left*$_i$) and energy (i.e., *energy_left*$_i$). Then, the time slack and the energy slack are distributed among jobs in the ready queue by the procedure

```
procedure DISTRIBUTE_SLACKS ($\mathcal{J}^{\text{ready}}$)    /* Let $\mathcal{J}^{\text{ready}} = \{J_{i_1}, J_{i_2}, \cdots, J_{i_n}\}$ where $i_1 < i_2 < \cdots, i_n$. */
1:  $available\_time_j := time\_slack_{i_j}$ for $1 \le j \le n$    /* the amount of time slack available for $J_{i_1}, J_{i_2}, \cdots, J_{i_j}$ */
2:  $total\_available\_time := available\_time_n$ , $total\_used\_time := 0$
3:  $\mathcal{J}^{\text{t}} := \{\}$    /* the set of jobs to which no more time can be allocated */
4:  $\mathcal{J}^{\text{w}} := \{\}$    /* the set of jobs with the full optional workloads */
5:  $num\_slices := C$    /* By adjusting the constant $C$, the tradeoff between the quality and the overhead can be explored. */
6:  for ($k := num\_slices$ to 1)    /* Distribute the time slack. */
7:      $time\_slice := total\_available\_time/k$
8:      let $J_{i_h} \in \mathcal{J}^{\text{ready}} - (\mathcal{J}^{\text{t}} \cup \mathcal{J}^{\text{w}})$ be the job with the lowest gradient $P'(speed_{i_h})/\rho'_{i_h}(workload_{i_h})$
9:      $\Delta t := \min\{time\_slice,\ available\_time_h,\ (u_{i_h} - workload_{i_h})/speed_{i_h}\}$
10:     if ($\Delta t \equiv available\_time_h$)
11:         $\mathcal{J}^{\text{t}} := \mathcal{J}^{\text{t}} \cup \{J_{i_h}\}$    /* $J_{i_h}$'s workload cannot be given more time, but its speed can be incremented in the loop below. */
12:     end if
13:     if ($\Delta t \equiv (u_{i_h} - workload_{i_h})/speed_{i_h}$)
14:         $\mathcal{J}^{\text{w}} := \mathcal{J}^{\text{w}} \cup \{J_{i_h}\}$    /* $J_{i_h}$'s workload cannot be incremented beyond $u_{i_h}$. */
15:     end if
16:     $available\_time_j \mathrel{-}= \Delta t$ for $h \le j \le n$
17:     $total\_available\_time \mathrel{-}= \Delta t$ , $total\_used\_time \mathrel{+}= \Delta t$
18:     $time\_left_{i_h} \mathrel{+}= \Delta t$ , $workload\_left_{i_h} \mathrel{+}= \Delta t \cdot speed_{i_h}$ , $workload_{i_h} \mathrel{+}= \Delta t \cdot speed_{i_h}$
19:     $energy\_left_{i_h} \mathrel{+}= \Delta t \cdot P(speed_{i_h})$ , $energy\_slack \mathrel{-}= \Delta t \cdot P(speed_{i_h})$
20: end for
21: $time\_slack_k := \max\{0,\ time\_slack_k - total\_used\_time\}$ for $1 \le k \le |\mathcal{J}|$
22: if ($energy\_slack > 0$)    /* Distribute the remaining energy slack by incrementing jobs' speeds. */
23:     for ($k := num\_slices$ to 1)
24:         let $J_i \in \mathcal{J}^{\text{ready}} - \mathcal{J}^{\text{w}}$ be the job with the lowest gradient $P'(speed_i)/\rho'_i(workload_i)$
25:         $\Delta E := energy\_slack/k$
26:         $\Delta s := P^{-1}(P(speed_i) + \Delta E/time\_left_i) - speed_i$
27:         if ($\Delta s > (u_i - workload\_left_i)/time\_left_i$)
28:             $\Delta s := (u_i - workload\_left_i)/time\_left_i$
29:             $\mathcal{J}^{\text{w}} := \mathcal{J}^{\text{w}} \cup \{J_i\}$    /* $J_i$'s workload cannot be incremented beyond $u_i$. */
30:         end if
31:         $speed_i \mathrel{+}= \Delta s$ , $workload\_left_i \mathrel{+}= \Delta s \cdot time\_left_i$ , $workload_i \mathrel{+}= \Delta s \cdot time\_left_i$
32:         $energy\_left_i \mathrel{+}= time\_left_i \cdot (P(speed_i + \Delta s) - P(speed_i))$
33:         $energy\_slack \mathrel{-}= time\_left_i \cdot (P(speed_i + \Delta s) - P(speed_i))$
34:     end for
35: else    /* When distributing the time slack, used more energy than is available. */
36:     Similarly, iteratively decrement the speed of the job with the highest gradient until $energy\_slack = 0$.
37: end if
end procedure
```

```
procedure STRETCH_TO_NTA ($J_k$)
    let $t$ and $NTA$ be the current time and the next arrival time, respectively.
    $\Delta t := \min\{NTA, d_k\} - t - time\_left_k$
    if ($\Delta t > 0$)
        $time\_slack_j := \max\{0,\ time\_slack_j - \Delta t\}$ for $1 \le j \le |\mathcal{J}|$
        $time\_left_k \mathrel{+}= \Delta t$
        $workload\_left_k^- := workload\_left_k$ , $workload_k^- := workload_k$ , $energy\_left_k^- := energy\_left_k$
        $workload\_left_k := \min\{u_k - (workload_i^- - workload\_left_i^-)\ ,\ time\_left_k \cdot P^{-1}(enery\_left_k^-/time\_left_k)\}$
        $workload_k := workload_k^- + (workload\_left_k - workload\_left_k^-)$
        $speed_k := workload\_left_k/time\_left_k$
        $energy\_left_k := time\_left_k \cdot P(speed_k)$
        $energy\_slack \mathrel{+}= (energy\_left_k^- - energy\_left_k)$    /* It can be easily checked that $energy\_left_k \le energy\_left_k^-$ . */
    end if
end procedure
```

Fig. 7 The algorithm to distribute the time slack and the energy slack

DISTRIBUTE_SLACKS; it gives a portion of the time slack to a job J_j in the ready queue by incrementing $time\_left_j$ and further allocates additional energy by incrementing $speed_j$, as well as updating the other state variables accordingly.

The variable *energy_slack* keeps track of the available energy that can be additionally used by jobs executing next. As in [6], the time slack consists of several levels of slacks, each of which corresponds to each different priority level. The slack in each priority level represents a cumulative value, i.e., the sum of the unused processor time left by the jobs with higher priorities. A job can utilize the unused time of completed higher-priority jobs and contributes to the lower priority slacks, which is called slack degradation [6]. For an EDF job set (e.g., obtained from a periodic EDF task set), the number of different priorities needed to completely reflect the preemption relationship can be as large as the number of jobs $|\mathcal{J}|$, while for a job set obtained from a periodic fixed-priority task set $\mathfrak{T}$, only $|\mathfrak{T}|$ levels of priorities are sufficient. For the time being, we describe a simple (but computationally expensive) implementation where $|\mathcal{J}|$ levels of slacks $\mathit{time\_slack}_i$ $(1 \leq i \leq |\mathcal{J}|)$ are managed, and then present an equivalent, low-overhead implementation (i.e., in Fig. 8). *energy_slack* and $\mathit{time\_slack}_i$ $(1 \leq i \leq |\mathcal{J}|)$ are initially set to zero, and updated according to the following slack management policy:

- When J_i finishes its execution, $\mathit{energy\_slack}_i$ is incremented by J_i's unused energy $\mathit{energy\_left}_i$. Furthermore, the lower-priority time slacks, i.e., $\mathit{time\_slack}_j$ $(i+1 \leq j \leq |\mathcal{J}|)$, are incremented by J_i's unused time $\mathit{time\_left}_i$. (See the procedure RECLAIM_SLACKS.)
- During the idle interval, $\mathit{time\_slack}_j$'s $(1 \leq j \leq |\mathcal{J}|)$ are decremented by the idle time, but not to below zero. (See the procedure DURING_IDLE_TIME.)
- When the time slack and the energy slack are distributed among jobs $J_{i_1}, J_{i_2}, \ldots, J_{i_n}$ $(i_1 < i_2 < \ldots, i_n)$ in the ready queue by incrementing $\mathit{time\_left}_{i_j}$ and $\mathit{energy\_left}_{i_j}$ by Δa_{i_j} and Δe_{i_j} $(1 \leq j \leq n)$, respectively, Δa_{i_j}'s and Δe_{i_j}'s should satisfy

$$\sum_{j=1}^{k} \Delta a_{i_j} \leq \mathit{time\_slack}_{i_k} \text{ for all } 1 \leq k \leq n \quad \text{and} \quad \sum_{j=1}^{n} \Delta e_{i_j} \leq \mathit{energy\_slack}. \tag{10}$$

  Then, *energy_slack* is decremented by $\sum_{j=1}^{n} \Delta e_j$ and $\mathit{time\_slack}_j$'s $(1 \leq j \leq |\mathcal{J}|)$ are decremented by $\sum_{j=1}^{n} \Delta a_j$, but not to below zero. (See the procedure DISTRIBUTE_SLACKS.)
- When J_i starts or resumes its execution, reset $\mathit{time\_slack}_j$ $(1 \leq j \leq i)$ to zero. (see the procedure INVALIDATE_*time*_SLACK.)

As in other algorithms which are based on the priority-based slack stealing method [1,6], we incorporate the method presented in [19] into the time-slack management policy; when there is only one job J_i ready for execution, J_i can exclusively use the processor until the closest event, i.e., d_i or the next release time of a job. (See the procedure STRETCH_TO_NTA.) This additional policy makes it possible for a job to use the lower priority time-slack, further improving the solution quality. The following lemma states that the on-line scheduling algorithm preserves deadline and energy constraints provided that the initial schedule obtained by an off-line algorithm meets those constraints. (Refer to [21] for the proof.)

Lemma 4.1. *Given a feasible off-line schedule* $\langle \mathbf{A}, \mathbf{o} \rangle = \langle (a_1, \ldots, a_{|\mathcal{J}|}), (o_1, \ldots, o_{|\mathcal{J}|}) \rangle$, *all the jobs meet their deadlines and* $E^{\text{on}}(t) \leq E^{\text{off}}(t)$ *for all* t *under the on-line scheduling algorithm in Figs. 6–7 where* $E^{\text{on}}(t)$ *is the cumulative energy consumption of the on-line schedule up to* t *and* $E^{\text{off}}(t)$ *is the total energy required to complete jobs released no later than* t *under the off-line schedule* $\langle \mathbf{A}, \mathbf{o} \rangle$, *i.e.,* $E^{\text{off}}(t) \overset{\text{def}}{=} \sum_{r_i \leq t} a_i \cdot P(o_i/a_i)$.

```
procedure INITIALIZE
    $\langle (a_1,\cdots,a_{|\mathcal{J}|}),(o_1,\cdots,o_{|\mathcal{J}|})\rangle :=$ an off-line job-level schedule /* A task-level schedule can be substituted. */
    $energy\_slack := 0$
    $time\_slack := \{\}$   /* can be efficiently implemented as a sorted list. */
end procedure
procedure INVALIDATE_TIME_SLACK ($J_i$)
    foreach ($time\_slack_j \in time\_slack$ such that $j \le i$)
        Remove $time\_slack_j$ from $time\_slack$.
    end foreach
end procedure
procedure DURING_IDLE_TIME ($\Delta t$)
    foreach ($time\_slack_j \in time\_slack$)
        $time\_slack_j \; -= \; \Delta t$
        If $time\_slack_j \le 0$, remove the variable $time\_slack_j$ from the set $time\_slack$.
    end foreach
end procedure
procedure RECLAIM_SLACKS ($J_i$)
    $energy\_slack \; += \; energy\_left_i$
    /* At this point, $\forall time\_slack_j \in time\_slack,\ j > i$ is guaranteed by INVALIDATE_TIME_SLACK. */
    foreach ($time\_slack_j \in time\_slack$)
        $time\_slack_j \; += \; time\_left_i$
    end for
    Insert into $time\_slack$ a new variable $time\_slack_{i+1}$ which is initially set to $time\_left_i$.
end procedure
procedure DISTRIBUTE_SLACKS ($\mathcal{J}^{\text{ready}}$)      /* Let $\mathcal{J}^{\text{ready}} = \{J_{i_1}, J_{i_2}, \cdots, J_{i_n}\}$ where $i_1 < i_2 < \cdots, i_n$. */
    for ($j := 1$ to $n$)
        let $k \le i_j$ be the largest index such that $time\_slack_k \in time\_slack$.
        $available\_time_j := time\_slack_k$   /* the amount of time slack available for $J_{i_1}, J_{i_2}, \cdots, J_{i_j}$ */
    end for
    identical to lines 2-20 of the algorithm in Figure 7.
    foreach ($time\_slack_j \in time\_slack$)
        $time\_slack_j \; -= \; total\_used\_time$
        If $time\_slack_j \le 0$, remove the variable $time\_slack_j$ from the set $time\_slack$.
    end for
    identical to lines 22-37 of the algorithm in Figure 7.
end procedure
```

Fig. 8 An efficient algorithm to manage the time slack

Figure 8 shows a low-overhead implementation of the time-slack management. The rationale behind the efficient implementation is that the number of different values in the time slack is much less than the number of priority levels and it is sufficient to keep track of different values only by dynamically creating and destroying program variables. In Fig. 8, the set *time_slack* keeps representative levels of the time slack, i.e., for a variable $time\_slack_i \in time\_slack$ at some time-instant t_0, the value of $time\_slack_i$ is the same as the value of the same variable in the original implementation (i.e., Figs. 6–7). Furthermore, if the variables in Figs. 6–7 satisfy at t_0

$$\forall 1 \le j < n \ , \ time\_slack_{i_j} = time\_slack_{i_j+1} = \cdots = time\_slack_{i_{j+1}-1} < time\_slack_{i_{j+1}}$$

where $1 = i_1 < i_2 < \cdots < i_n = |\mathcal{J}|$, the set *time_slack* must keep the variables $time\_slack_{i_j}$ $(1 \le j \le n)$ at t_0. Using these properties, it can be easily checked that the new implementation is equivalent to the original implementation. We now derive the upper bounds on the size of the set *time_slack*. (Refer to [21] for the proof.)

Lemma 4.2. *Let K be the length of the longest path in the directed graph $G\langle V, E\rangle$ given by*

$$V = \{v_1, v_2, \ldots, v_{|\mathcal{J}|}\} \quad and \quad E = \{(v_i, v_j) \mid i < j \ \wedge \ r_i > r_j\}. \tag{11}$$

Then, the number of variables in time_slack is always no larger than $K + 1$.

Especially, the upper bound for a job set obtained from a periodic task set can be easily obtained:

Lemma 4.3. *Let $\mathcal{J}$ be a job set obtained from a periodic EDF task set $\mathfrak{T} = \{\tau_1, \ldots, \tau_{|\mathfrak{T}|}\}$ Then, the length of the longest path in the directed graph $G\langle V, E\rangle$ defined by Eq. (11) is no larger than $|\mathfrak{T}| - 1$.*

Proof: Without loss of generality, assume that $\tau_1, \tau_2, \ldots, \tau_{|\mathfrak{T}|}$ are sorted in an increasing order of the lengths of their relative deadlines. Note that $(v_i, v_j) \in E$ implies $(d_i - r_i) < (d_j - r_j)$, i.e., J_i and J_j are instances of tasks $\tau_{i'}$ and $\tau_{j'}$, respectively, such that $1 \le i' < j' \le |\mathfrak{T}|$. Therefore, the length of any path in $G\langle V, E\rangle$ is bounded by $|\mathfrak{T}| - 1$. □

5 Experimental results

In order to evaluate the performance of the proposed algorithms, we performed several experiments with a test job set constructed from a periodic task set, whose timing parameters are given in Table 1. (The task set is borrowed from [2].) The task set consists of 11 flexible periodic tasks whose maximum utilization (mandatory + optional) utilization is 2.3 assuming the uniform processor speed of 1. The test job set consists of 393 task instances. (One hyperperiod of the task set in Table 1 has 393 task instances.) Each job (i.e., task instance) is associated with three types of reward functions: exponential (i.e., $\rho_i^{\exp}(x)$), logarithmic (i.e., $\rho_i^{\log}(x)$), and linear (i.e., $\rho_i^{\text{linear}}(x)$) functions. In our experiments, we assumed that the energy consumption (per CPU cycles) is quadratically dependent on the processor speed. That is, the instantaneous power consumption (per time) is cubically dependent on the processor speed; $P(t) = \mathcal{S}(t)^3$.

First, we implemented the polynomial-time optimal off-line algorithm described in Section 3, and collected the maximum achievable total reward value for each energy budget value using the optimal off-line algorithm. Figure 9 showsthe Pareto-optimal curve for the amount of available energy and the achievable total reward for the test job set. The energy and reward values were normalized over the maximum achievable total reward (i.e., $F(\mathbf{u})$) and the energy consumption for the job set with the maximum optional workload (i.e., $E(\langle \mathbf{A}[\mathbf{u}], \mathbf{u}\rangle)$),

Table 1 A flexible periodic task set (borrowed from [2])

Periodic task	Period	Relative deadline	Hyperperiod /Period	Mandatory workload	Max. opt. workload	Reward functions (x: opt. workload) $\rho^{\exp}(x)$	$\rho^{\log}(x)$	$\rho^{\text{linear}}(x)$
τ_1	20	10	108	4	6	$15(1-e^{-x})$	$7\ln(20x+1)$	$5x$
τ_2	30	25	72	6	12	$20(1-e^{-3x})$	$10\ln(50x+1)$	$7x$
τ_3	40	30	54	2	3	$4(1-e^{-x})$	$2\ln(10x+1)$	$2x$
τ_4	60	30	36	1	1	$10(1-e^{-x/2})$	$5\ln(5x+1)$	$4x$
τ_5	60	50	36	1	1	$10(1-e^{-x/5})$	$5\ln(25x+1)$	$4x$
τ_6	80	40	27	6	6	$5(1-e^{-x})$	$3\ln(30x+1)$	$2x$
τ_7	90	60	24	3	15	$17(1-e^{-x})$	$8\ln(8x+1)$	$6x$
τ_8	120	90	18	12	3	$8(1-e^{-x})$	$4\ln(6x+1)$	$3x$
τ_9	240	160	9	9	19	$8(1-e^{-x})$	$4\ln(9x+1)$	$3x$
τ_{10}	270	180	8	40	20	$12(1-e^{-x/2})$	$6\ln(12x+1)$	$5x$
τ_{11}	2160	1890	1	120	180	$5(1-e^{-x})$	$3\ln(15x+1)$	$2x$

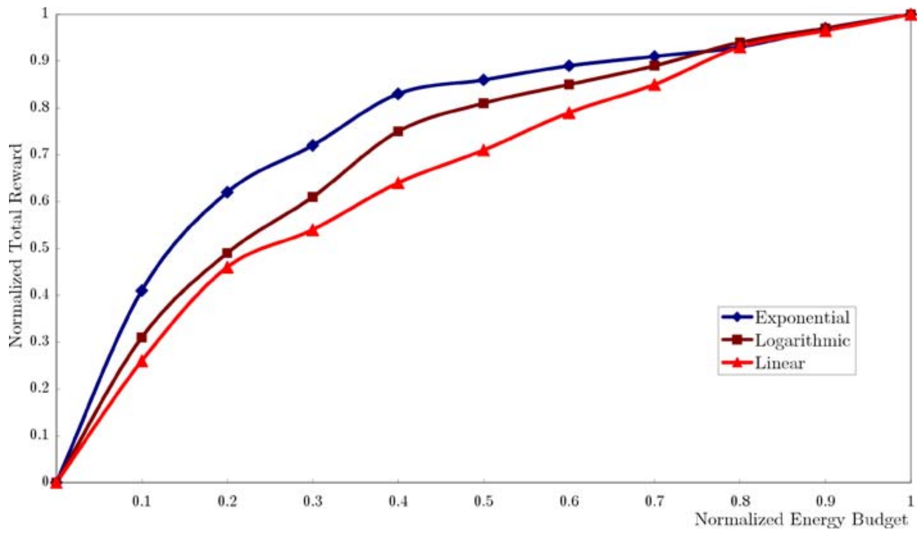

Fig. 9 Pareto-optimal curve for energy and reward obtained by the optimal off-line scheduling algorithm

respectively. As shown in Fig. 9, the achievable total reward is a nearly concave function of the energy budget for each type of reward functions. The rate of the decrease in the total reward is very slow around the maximum energy budget; for exponential reward functions, even with 70% of the maximum energy budget, more than 90% of the maximum total reward can be achieved.

Next, we evaluated the energy-reward performance of the on-line algorithm in Section 4. For a comparison, the optimal off-line algorithm computes the theoretical lower bound with the complete execution trace information. In each experiment, the actual mandatory workload and the upper bound of the optional workload of each job was randomly drawn from a uniform distribution within the range of [n·WCET/10,WCET] of each task for $1 \le n \le 9$. The energy budget E_{budget} was set to be $E(\langle \mathbf{A}[\mathbf{u}], \mathbf{u}\rangle)/2$. Results were normalized over the total reward of each job set scheduled by the off-line algorithm. Figure 10 showsthe relative performance of

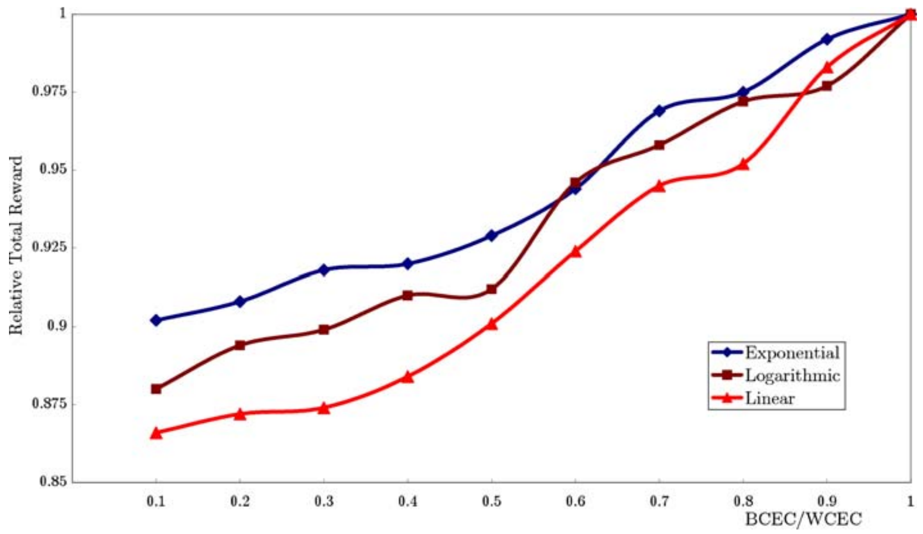

Fig. 10 Evaluation of on-line scheduling algorithm with the varying degree of workload fluctuation

on-line algorithm. For experiments, initial off-line schedules were computed by the off-line scheduling algorithm in 4. As shown in Fig. 10, the on-line algorithm is sufficiently efficient; the result obtained by the on-line algorithm is, even in the worst case (i.e., BCET/WCET $= 0.1$), only 14% worse than the theoretical lower bound.

6 Conclusion

We investigated the problem of reward-based voltage scheduling for the general task model where each job has its own release time and deadline. With the increasing importance of battery-operated embedded systems and flexible applications, considerable research efforts have been made on both voltage scheduling and reward-based scheduling. However, the combined scheduling problem of maximizing the total reward subject to energy constraints has been relatively unexplored.

First, we present a polynomial-time optimal off-line algorithm for the problem. In order to search the complicated solution space efficiently, we exploit properties of energy-optimal voltage schedules. Second, we propose a low-overhead on-line algorithm based on the observations from the optimal off-line algorithm. Despite its simplicity, the on-line algorithm is sufficiently efficient in terms of energy-reward performance. Experimental results show that the quality of solution computed by the on-line algorithm is no more than 14% worse than that of the optimal off-line solution.

The proposed algorithms can be further extended in several directions. As our immediate future work, we are interested in a more realistic processor model with static energy consumption and transition overheads in time and energy and . Furthermore, In addition, we plan to develop off-line and on-line algorithms for fixed-priority real-time systems.

References

1. Aydin, H., R. Melhem, D. Mossé, and P.M. Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proc. of Real-Time Systems Symposium*, 2001.

2. Aydin, H., R. Melhem, D. Mossé, and P.M. Alvarez. Optimal Reward-Based Scheduling for Periodic Real-Time Tasks. *IEEE Transactions on Computers*, 50(2),111–130, 2001.
3. Ben-Tal, A. and A. Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. SIAM, 2001.
4. Chung, J.-Y., J.W.-S. Liu, and K.-J. Lin. Scheduling Periodic Jobs that Allow Imprecise Results. *IEEE Transactions on Computers*, 39(9),1156–1173, 1990.
5. Dey, J.K., J. Kurose, and D. Towsley. On-Line Scheduling Policies for a Class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks. *IEEE Transactions on Computers*, 45(7),802–813, 1996.
6. Gruian, F. Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors. In *Proc. of International Symposium on Low Power Electronics and Design*, pp. 46–51, 2001.
7. Hong, I., G. Qu, M. Potkonjak, and M.B. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In *Proc. of Real-Time Systems Symposium*, pp. 178–187, 1998.
8. Kim, W., J. Kim, and S.L. Min. A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis. In *Proc. of Design, Automation and Test in Europe*, 2002.
9. Kim, W., J. Kim, and S.L. Min. Dynamic Voltage Scaling Algorithm for Fixed-Priority Real-Time Systems Using Work-Demand Analysis. In *Proc. of International Symposium On Low Power Electronics and Design*, 2003.
10. Kim, W., 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. of Real-Time and Embedded Technology and Applications Symposium*, 2002.
11. Kwon, W.-C. and T. Kim. Optimal Voltage Allocation Techniques for Dynamically Variable Voltage Processors. In *Proc. of Design Automation Conference*, pp. 125–130, 2003.
12. Lin, K.-J., S. Natarajan, and J.W.-S. Liu. Imprecise Results: Utilizing Partial Computations in Real-Time Systems. In *Proc. of Real-Time Systems Symposium*, pp. 210–217, 1987.
13. Liu, W.-S. *Real-Time Systems*. Prentice Hall, 2000.
14. Pillai, P. and K.G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proc. of ACM Symposium on Operating Systems Principles*, 2001.
15. Rusu, C., R. Melhem, and D. Mossé. Maximizing the System Value While Satisfying Time and Energy Constraints. In *Proc. of Real-Time Systems Symposium*, pp. 246–255, 2002.
16. Rusu, C., R. Melhem, and D. Mossé. Maximizing Rewards for Real-Time Applications with Energy Constraints. *ACM Transactions on Embedded Computing Systems*, 2(4), 537–559, 2003.
17. Sakurai, T. and A. Newton. Alpha-power Law MOSFET Model and Its Application to CMOS Inverter Delay and Other Formulars. *IEEE Journal of Solid State Circuits*, 25(2), 584–594, 1990.
18. Shih, W.-K., J.W.-S. Liu, and J.-Y. Chung. Algorithms for Scheduling Imprecise Computations with Timing Constraints. *SIAM Journal on Computing*, 20(3), 537–552, 1991.
19. Shin, Y. and K. Choi. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In *Proc. of Design Automatioin Conference*, pp. 134–139, 1999.
20. Yao, F., A. Demers, and S. Shenker. A Scheduling Model for Reduced CPU Energy. In *Proc. of IEEE Annual Foundations of Computer Science*, pp. 374–382, 1995.
21. Yun, H.-S. and J. Kim. Reward-Based Voltage Scheduling for Dynamic-Priority Hard Real-Time Systems. Technical report. Available at *http://davinci.snu.ac.kr/Download/daem_techrep.pdf*.
22. Yun, H.-S. and J. Kim. On Energy-Optimal Voltage Scheduling for Fixed-Priority Hard Real-Time Systems. *ACM Transactions on Embedded Computing Systems*, 2(3), 393–430, 2003.