Reward-Based Voltage Scheduling for Hard Real-Time Systems with Energy Constraints*

Han-Saem Yun and Jihong Kim

School of Computer Science and Engineering
Seoul National University, Seoul 151-742, Korea
{hsyun, jihong@davinci.snu.ac.kr}

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 for the general task model. We present an *optimal off-line* algorithm and an efficient *on-line* algorithm for jobs with their own release-times/deadlines. Our work is the first significant result for the general task model.

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 executed before the deadline. The optional part is assumed to follow the mandatory part in sequence and can be interrupted at any time. A non-decreasing concave 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

* This research was supported by University IT Research Center Project

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 being obtained by adding one more dimension to the solution space of either the reward-based scheduling problem or the voltage scheduling problem; 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 involves two-dimensional objectives, maximizing the total reward (from the reward-based scheduling) and minimizing the energy consumption (from the 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 re-

ward 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 [21]. 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, 21] 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 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 leverage the workload variation to increase the reward within energy budget. Experimental results show that the on-line algorithm is sufficiently efficient; the quality of so-

[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.

lution (i.e., the total reward) computed by the on-line algorithm is only $3 \sim 13\%$ 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 Section 3 while 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 $\mathcal{J} = \{J_1, J_2, \cdots, J_{|\mathcal{J}|}\}$ 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. $[m_i, u_i]$).)
- f_i: the reward given as a function of the total workload.

The job model can be directly applicable to a periodic real-time system by considering all the task instances within the hyperperiod. We assume that the job set $\mathcal{J}$ follows the EDF priority as in [20].[2] Note that, 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 $\mathbf{o} = \{o_1, o_2, \cdots, o_{|\mathcal{J}|}\}$ by the workload of J_i and the workload tuple, respectively. Associated with each optional workload o_i is a reward function $f_i(o_i)$, which is assume to be nondecreasing, concave, and continuously differentiable over the interval $[m_i, u_i]$ as in [2, 16]. The derivative of f_i is denoted by f_i'.

Given a workload tuple $\mathbf{o} = \{o_1, o_2, \cdots, o_{|\mathcal{J}|}\}$, the total reward F, our optimization goal, is given by $F(\mathbf{o}) = \sum_{i=1}^{|\mathcal{J}|} f_i(o_i)$. 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. Given two workload tuples $\mathbf{o}_1$ and $\mathbf{o}_2$, we write $\mathbf{o}_1 \le \mathbf{o}_2$ if $o_i[\mathbf{o}_1] \le o_i[\mathbf{o}_2]$ for all $1 \le i \le |\mathcal{J}|$. Note that, for such $\mathbf{o}_1$ and $\mathbf{o}_2$, $F(\mathbf{o}_1) \le F(\mathbf{o}_2)$. Particularly, we use $\mathbf{m}$ and $\mathbf{u}$ to denote $(m_1, m_2, \cdots, m_{|\mathcal{J}|})$ and $(u_1, u_2, \cdots, u_{|\mathcal{J}|})$, respectively. Note that $\mathbf{m}$ and $\mathbf{u}$ is the lower and upper bounds for $\mathbf{o}$. The solution space given by $\mathbf{o}$ is written by O , i.e., $O = \{\mathbf{o} \mid \mathbf{m} \le \mathbf{o} \le \mathbf{u}\}$. For $\mathbf{o}_1$ and $\mathbf{o}_2$, $\mathbf{o}_1 - \mathbf{o}_2$ is defined by $o_i[\mathbf{o}_1 - \mathbf{o}_2] = o_i[\mathbf{o}_1] - o_i[\mathbf{o}_2]$ $(1 \le i \le |\mathcal{J}|)$.

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. (The exact condition for the feasibility is explained later.) As with other related

[2] 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 $d_j \le r_i$.

work [20, 21], we assume that the processor speed can be varied continuously[3] with a negligible overhead both in time and power. Furthermore, we model that the power P, energy consumed per unit time, is a 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 is dependent only on the processor speed.

The energy-optimal schedule for $\mathbf{o}$ is defined to be a schedule $\mathcal{S}(t)$ feasible for $\mathbf{o}$ that minimizes $E(\mathcal{S}) = \int_{t_s}^{t_f} P(\mathcal{S}(t))\, dt$ 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 $\mathcal{S}[\mathbf{o}]$, 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, 21], we can easily establish a one-to-one correspondence between $\mathcal{S}(t)$ and the the allowed execution time a_i allocated to each $J_i \in \mathcal{J}$. Given a feasible voltage schedule $\mathcal{S}$, the corresponding tuple of the allowed execution times $(a_1, a_2, \cdots, a_{|\mathcal{J}|})$ is uniquely determined. Conversely, given $\mathbf{A} = (a_1, a_2, \cdots, a_{|\mathcal{J}|})$, the corresponding offline voltage schedule $\mathcal{S}_{\mathbf{A}}$ can be uniquely constructed by assigning the constant execution speed o_i/a_i to J_i. $\mathbf{A}$ is said to be *feasible* if the corresponding voltage schedule $\mathcal{S}_{\mathbf{A}}$ is feasible. For an EDF job set $\mathcal{J}$, $\mathbf{A} = (a_1, a_2, \cdots, a_{|\mathcal{J}|})$ is feasible if and only if the following condition is satisfied (See [21] for a proof.):

Condition I (EDF Feasibility Condition).

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

The energy consumption of the voltage schedule in terms of $\mathbf{A}$ is given by $E(\mathbf{A}) = \sum_{i=1}^{|\mathcal{J}|} a_i \cdot P(o_i/a_i)$. For a fixed workload tuple $\mathbf{o}$, the energy-optimal voltage scheduling problem is stated as maximizing $E(\mathbf{A})$ subject to Condition I, and can be optimally solved by Yao's algorithm. Given a workload tuple $\mathbf{o}$, the corresponding energy-optimal voltage schedule in terms of $\mathbf{A}$ is denoted by $\mathbf{A}[\mathbf{o}] = (a_1[\mathbf{o}], a_2[\mathbf{o}], \cdots, a_{|\mathcal{J}|}[\mathbf{o}])$.[4]

Now, the reward-based voltage scheduling problem is formulated as follows:

Find a workload tuple $\mathbf{o} \in O$ such that the total reward $F(\mathbf{o})$ is maximized subject to $E(\mathbf{A}[\mathbf{o}]) \le E_{\text{budget}}$.

The main source of difficulty is that $\mathbf{A}[\mathbf{o}]$ is not explicitly represented in terms of $\mathbf{o} = \{o_1, o_2, \cdots, o_{|\mathcal{J}|}\}$, which makes it difficult to explore the solution space given by $\mathbf{o}$. In Section 3, we present how to efficiently search the optimal solution of the reward-based voltage scheduling problem by exploiting the properties of energy-optimal voltage schedules.

[3] The results in this paper can be easily extended to a processor model with a limited number of voltage levels using the result of [11].

[4] 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 $\mathbf{o}$.

3 Optimal Off-Line Algorithm

In this section, we present an optimal off-line algorithm for the problem. Before describing the algorithm, since the solution space given by $\mathbf{o}$ is implicitly represented by a condition in which its corresponding energy-optimal voltage schedule is involved (i.e., $E(\mathbf{A}[\mathbf{o}]) \leq E_{\text{budget}}$), we characterize some useful properties of energy-optimal voltage schedules, which provide a basis of later proofs.

3.1 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 $\mathcal{S}[\mathbf{o}]$ (equivalently, $\mathbf{A}[\mathbf{o}]$). $s_i[\mathbf{o}]$ and $I_i[\mathbf{o}]$ denote the (constant) speed of J_i and the union of intervals in which J_i is executed under $\mathcal{S}[\mathbf{o}]$, 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 $P'(s_i[\mathbf{o}])/f_i'(o_i[\mathbf{o}])$ where P' is the derivative of P, and is called the *gradient* of J_i .

$\mathcal{J}_k[\mathbf{o}]$ represents the set of jobs scheduled at the i-th iteration in 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 non-increasing 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 relation $\sim_{[\mathbf{o}]}$ on $\mathcal{J}$ is defined by

$$J_i \sim_{[\mathbf{o}]} J_j \quad \text{iff} \quad \exists\, k,\ J_i, J_j \in \mathcal{J}_k[\mathbf{o}] \ .$$

The relation $\sim_{[\mathbf{o}]}$ is an equivalence relation and gives rise to a partition of $\mathcal{J}$, written $\mathcal{G}[\mathbf{o}]$, i.e., $\mathcal{G}[\mathbf{o}] = \{\mathcal{J}_k[\mathbf{o}] \mid k = 1, 2, \cdots\}$. In Figure 1, Yao's algorithm is described by the symbols defined so far.

$\mathcal{J}_k[\mathbf{o}]$ 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.,

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

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

$$\begin{aligned} \rho_k[\mathbf{o}] &\stackrel{\text{def}}{=} \min\{\, f_i'(o_i[\mathbf{o}]) \mid J_i \in \mathcal{J}_k^+[\mathbf{o}] \,\} \\ \nabla_k[\mathbf{o}] &\stackrel{\text{def}}{=} \max\{g_i[\mathbf{o}] \mid J_i \in \mathcal{J}_k^+[\mathbf{o}] \,\} \ \equiv\ P'(\sigma_k[\mathbf{o}])/\rho_k[\mathbf{o}] \ . \end{aligned}$$

When $\mathcal{J}_k^+[\mathbf{o}]$ is empty, $\rho_k[\mathbf{o}]$ and $\nabla_k[\mathbf{o}]$ are set to be ∞ and 0 . $\nabla_k[\mathbf{o}]$ is called the *gradient* of $\mathcal{J}_k[\mathbf{o}]$. $\nabla[\mathbf{o}]$ denotes the largest one among $\nabla_1[\mathbf{o}], \nabla_2[\mathbf{o}], \cdots$, i.e.,

$$\nabla[\mathbf{o}] \stackrel{\text{def}}{=} \max\{\nabla_k[\mathbf{o}] \mid k = 1, 2, \cdots\} \ \equiv\ \max\{g_i[\mathbf{o}] \mid J_i \in \cup_{k \geq 1} \mathcal{J}_k^+[\mathbf{o}] \,\} \ .$$

procedure ENERGY_OPTIMAL_VOLTAGE_SCHEDULING($\mathcal{J}$, $\mathbf{o}$)

$\mathcal{J}' := \mathcal{J}$
$\mathbf{I} := \cup_{i=1}^{|\mathcal{J}|} [r_i, d_i]$ /* the whole execution interval */
$k := 1$ /* the iteration number */
while ($\mathcal{J}' \neq \emptyset$)
  $\mathcal{T}_{\mathcal{J}'} := \{r_i, d_i | J_i \in \mathcal{J}'\}$
  Find $r, d \in \mathcal{T}_{\mathcal{J}'}$ $(r < d)$ such that $\sigma_{[r,d]} = \dfrac{\sum_{i/[r_i,d_i] \subseteq [r,d]} o_i[\mathbf{o}]}{\|\mathbf{I} \cap [r,d]\|}$ is maximized.
  /* Ties are broken by preferring the largest $d - r$ and then the smallest r. */
  $\mathcal{J}_k[\mathbf{o}] := \{J_i | [r_i, d_i] \subseteq [r,d]\}$ /* the set of jobs scheduled at the k-th iteration */
  $\sigma_k[\mathbf{o}] := \sigma_{[r,d]}$ /* the constant speed allocated to jobs in $\mathcal{J}_k[\mathbf{o}]$ */
  foreach ($J_i \in \mathcal{J}_k[\mathbf{o}]$)
    $s_i[\mathbf{o}] := \sigma_k[\mathbf{o}]$ /* the constant speed allocated to J_i */
    $a_i[\mathbf{o}] := o_i[\mathbf{o}]/s_i[\mathbf{o}]$ /* the execution time allocated to J_i */
    $I_i[\mathbf{o}] :=$ the union of intervals in which J_i is executed under the EDF policy
  end foreach
  $\mathcal{J}' := \mathcal{J}' - \mathcal{J}_k[\mathbf{o}]$
  $\mathbf{I} := \mathbf{I} - [r,d]$
  $k := k + 1$
end while
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. 1. Yao's algorithm to compute an energy-optimal voltage schedule.

$\mathcal{J}_k^{\top}[\mathbf{o}]$ represents the subset of $\mathcal{J}_k^{+}[\mathbf{o}]$ that consists of jobs with the smallest f' value (equivalently, the largest gradient), i.e.,

$$\mathcal{J}_k^{\top}[\mathbf{o}] \stackrel{\text{def}}{=} \{J_i \in \mathcal{J}_k^{+}[\mathbf{o}] \mid f_i'(o_i[\mathbf{o}]) = \rho_k[\mathbf{o}]\} \equiv \{J_i \in \mathcal{J}_k^{+}[\mathbf{o}] \mid g_i[\mathbf{o}] = \nabla_k[\mathbf{o}]\} .$$

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.,

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

For workload tuples $\mathbf{o}_1$ and $\mathbf{o}_2$, we write $\mathbf{o}_1 \approx \mathbf{o}_2$ if $\mathcal{G}[\mathbf{o}_1] \equiv \mathcal{G}[\mathbf{o}_2]$.[5] 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 points. Because $\mathbf{I}_{it_i[\mathbf{o}_1]}[\mathbf{o}_1] \equiv \mathbf{I}_{it_i[\mathbf{o}_2]}[\mathbf{o}_2]$ for a job $J_i \in \mathcal{J}$, $\mathbf{A}[\mathbf{o}_2] = (a_1[\mathbf{o}_2], \cdots, a_{|\mathcal{J}|}[\mathbf{o}_2])$ can be expressed in terms of $\mathbf{o}_2$ and $\mathbf{A}[\mathbf{o}_1] = (a_1[\mathbf{o}_1], \cdots, a_{|\mathcal{J}|}[\mathbf{o}_1])$ as follows.

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

The relation $\approx$ is an equivalence relation and forms an (infinite) partition of O . Generally, $\mathbf{A}[\mathbf{o}]$ is not explicitly represented in terms of $\mathbf{o}$. However, if $\mathbf{A}[\mathbf{o}']$ is available for

[5] $\mathcal{G}[\mathbf{o}_1] \equiv \mathcal{G}[\mathbf{o}_2]$ 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 $k \geq 1$.

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 3.2 exploits this property in searching the optimal solution.

Now, we describe useful properties that characterize the relation between some attributes of energy-optimal voltage schedules for different workload tuples. The monotonicity of the speed $s_i[\mathbf{o}]$ and and the gradient $g_i[\mathbf{o}]$ of a job J_i with respect to the workload tuple $\mathbf{o}$ is stated in the following lemma.

Lemma 1. *For* $\mathbf{o}$ *and* $\mathbf{o}'$ *such that* $\mathbf{o} \leq \mathbf{o}'$,

$$s_i[\mathbf{o}] \leq s_i[\mathbf{o}'] \quad \textit{and} \quad g_i[\mathbf{o}] \leq g_i[\mathbf{o}'] \quad \textit{for all } 1 \leq i \leq |\mathcal{J}| \ .$$

Proof. For $1 \leq j \leq |\mathcal{J}|$, let $\mathbf{o}_j$ be defined by

$$o_h[\mathbf{o}_j] = \begin{cases} o_h[\mathbf{o}] & h \leq j \\ o_h[\mathbf{o}'] & h > j \ . \end{cases}$$

It is easy to verify that, for $\mathbf{o}_j$ and $\mathbf{o}_{j+1}$ $(1 \leq j < |\mathcal{J}|)$,

$$\begin{aligned} &\forall J_i \in \mathcal{J} \text{ s.t. } it_i[\mathbf{o}_j] > it_{j+1}[\mathbf{o}_j] \ , \ s_i[\mathbf{o}_{j+1}] = s_i[\mathbf{o}_j] \quad \text{and} \\ &\forall J_i \in \mathcal{J} \text{ s.t. } it_i[\mathbf{o}_j] \leq it_{j+1}[\mathbf{o}_j] \ , \ s_i[\mathbf{o}_{j+1}] \geq s_i[\mathbf{o}_j] \ . \end{aligned}$$

It immediately follows that

$$s_i[\mathbf{o}] = s_i[\mathbf{o}_1] \leq s_i[\mathbf{o}_2] \leq \cdots \leq s_i[\mathbf{o}_{|\mathcal{J}|}] = s_i[\mathbf{o}'] \ .$$

Furthermore, from the convexity of P and the concavity of f, we have

$$g_i[\mathbf{o}] = \frac{P'(s_i[\mathbf{o}])}{f_i'(o_i[\mathbf{o}])} \leq \frac{P'(s_i[\mathbf{o}'])}{f_i'(o_i[\mathbf{o}])} \leq \frac{P'(s_i[\mathbf{o}'])}{f_i'(o_i[\mathbf{o}'])} = g_i[\mathbf{o}'] \ . \qquad \square$$

Note that the converse of Lemma 1 does not always hold, i.e., $s_i[\mathbf{o}] \leq s_i[\mathbf{o}']$ $(1 \leq i \leq |\mathcal{J}|)$ does not imply $\mathbf{o} \leq \mathbf{o}'$. However, it preserves the order in $\mathcal{S}[\mathbf{o}]$ as follows.

Lemma 2. *For* $\mathbf{o}$ *and* $\mathbf{o}'$ *such that*

$$\begin{aligned} &\forall\, 1 \leq i \leq |\mathcal{J}| \ , \ \ s_i[\mathbf{o}] \leq s_i[\mathbf{o}'] \ , \\ &\forall\, t \ , \ \mathcal{S}[\mathbf{o}](t) \leq \mathcal{S}[\mathbf{o}'](t) \quad \textit{and} \quad E(\mathcal{S}[\mathbf{o}]) \leq E(\mathcal{S}[\mathbf{o}']) \ . \end{aligned}$$

Proof. To find the explicit representation of $\mathcal{S}[\mathbf{o}](t)$ (resp. $\mathcal{S}[\mathbf{o}'](t)$) in terms of $s_i[\mathbf{o}]$ (resp. $s_i[\mathbf{o}']$), we first prove that

$$\forall\, 1 \leq i \leq |\mathcal{J}| \ , \ \forall\, r_i \leq t \leq d_i \ , \ \mathcal{S}[\mathbf{o}](t) \geq s_i[\mathbf{o}] \ . \tag{2}$$

Suppose to the contrary that for some i , s_0 , t_0 and Δt (where $r_i \leq t_0 < t_0 + \Delta t \leq d_i$), we have $\forall\, t_0 \leq t \leq t_0 + \Delta t, \ \mathcal{S}[\mathbf{o}](t) = s_0 < s_i[\mathbf{o}]$. Let $\mathcal{S}(t)$ be defined by

$$\mathcal{S}(t) = \begin{cases} \dfrac{o_i[\mathbf{o}] + s_o \cdot \Delta t}{a_i[\mathbf{o}] + \Delta t} \ \ (< s_i[\mathbf{o}]) & t_0 \leq t \leq t_0 + \Delta t \ \ \vee \ \ t \in I_i[\mathbf{o}] \ , \\ \mathcal{S}[\mathbf{o}](t) & \text{otherwise.} \end{cases}$$

Then, it is easy to check that $\mathcal{S}(t)$ is feasible for $\mathbf{o}$ and $E(\mathcal{S}) < E(\mathcal{S}[\mathbf{o}])$. Thus, $\mathcal{S}[\mathbf{o}]$ is not the energy-optimal voltage schedule for $\mathbf{o}$, a contradiction.
Furthermore, from the definition of $s_i[\mathbf{o}]$, we have

$$\mathcal{S}[\mathbf{o}](t) \in \{s_i[\mathbf{o}] \mid r_i \leq t \leq d_i\} . \tag{3}$$

From Eq.(2) and Eq.(3), $\mathcal{S}[\mathbf{o}](t)$ can be expressed in terms of $s_i[\mathbf{o}]$ as follows:

$$\mathcal{S}[\mathbf{o}](t) = \max\{s_i[\mathbf{o}] \mid r_i \leq t \leq d_i\} .$$

Similarly, $\mathcal{S}[\mathbf{o}']](t)$ is given by

$$\mathcal{S}[\mathbf{o}'](t) = \max\{s_i[\mathbf{o}'] \mid r_i \leq t \leq d_i\} \geq \max\{s_i[\mathbf{o}] \mid r_i \leq t \leq d_i\} = \mathcal{S}[\mathbf{o}](t) .$$

It immediately follows that $E(\mathcal{S}[\mathbf{o}]) \leq E(\mathcal{S}[\mathbf{o}'])$. □

3.2 The Optimal Algorithm

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]. 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 3.1. 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 Figure 2 describes the overall processing steps of our algorithm. The search is guided by the *global gradient* g . Initially, g is initially set to $\nabla[\mathbf{u}]$, 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_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 (Figure 3) such that each $g_i[\mathbf{o}\langle g\rangle]$ does not exceed the decreased g. (Note that $g_i[\mathbf{o}]$ decreases with $\mathbf{o}$ by Lemma 1.)

procedure MAXIMIZE_REWARD

$E := E(\mathcal{S}[\mathbf{u}])$

$g := \max\{g_i[\mathbf{u}] | J_i \in \cup_{k \geq 1} \mathcal{J}_k^+[\mathbf{u}]\} \quad (= \nabla[\mathbf{u}])$

$\mathbf{o}\langle g \rangle := \mathbf{u}$

while $(E > E_{\text{budget}})$

  $\mathbf{o} := \mathbf{o}\langle g \rangle$

  $g_{\text{s}} := g$

  $g :=$ NEXT_GRADIENT$(\mathbf{o}\langle g \rangle, g)$

  $g := \max(\{g\} \cup \{\nabla_{k'}[\mathbf{o}] \mid \nabla_{k'}[\mathbf{o}] < \nabla_k[\mathbf{o}]\})$

  $\mathbf{o}\langle g \rangle :=$ DECREASE_WORKLOADS$(\mathbf{o}\langle g \rangle, g)$

  $E := E(\mathcal{S}[\mathbf{o}\langle g \rangle])$

end while

/* The optimal solution is between $\mathbf{o}\langle g \rangle$ and $\mathbf{o}$. */

Solve the simultaneous equations given by

Eq.(7) for all $\mathcal{J}_k[\mathbf{o}] \in \mathcal{G}\langle g_{\text{s}} \rangle[\mathbf{o}]$ and

$$\sum_{\mathcal{J}_k[\mathbf{o}] \in \mathcal{G}\langle g_{\text{s}} \rangle[\mathbf{o}]} P\left(\frac{\sum_{J_i \in \mathcal{J}_k[\mathbf{o}]} \gamma_i(h_k)}{\|\mathbf{I}_k\|}\right) \cdot \|\mathbf{I}_k\| \;+\; \sum_{\mathcal{J}_k[\mathbf{o}] \in \mathcal{G}[\mathbf{o}] - \mathcal{G}\langle g_{\text{s}} \rangle[\mathbf{o}]} P\left(\frac{\sum_{J_i \in \mathcal{J}_k[\mathbf{o}]} o_i[\mathbf{o}]}{\|\mathbf{I}_k\|}\right) \cdot \|\mathbf{I}_k\| \;=\; E_{\text{budget}} \; .$$

foreach $(J_i \in \mathcal{J})$

  if $(\mathcal{J}_{it_i[\mathbf{o}]} \in \mathcal{G}\langle g_{\text{s}} \rangle[\mathbf{o}])$

    $o_i := \gamma_i(h_{it_i[\mathbf{o}]})$

  else

    $o_i := o_i[\mathbf{o}]$

  end if

end foreach

/* $E(\mathcal{S}[(o_1, o_2, \cdots, o_{|\mathcal{J}|})]) = E_{\max}$ and $\sum_{i=1}^{|\mathcal{J}|} f(o_i)$ is maximum. */

return $(o_1, o_2, \cdots, o_{|\mathcal{J}|})$

end procedure

Fig. 2. The optimal off-line reward-based voltage scheduling algorithm.

After each invoke 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 sufficient small number and Δo_i's (> 0) satisfy

$$g_i[\mathbf{o}\langle g - \Delta g \rangle] \;=\; g - \Delta g \quad \text{for all } i \text{ s.t. } g_i[\mathbf{o}\langle g \rangle] \geq g \;\wedge\; o_i[\mathbf{o}\langle g \rangle] > m_i \;. \tag{4}$$

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.(4). Informally, $\Delta\mathbf{o} = (\Delta o_1, \Delta o_2, \cdots, \Delta o_{|\mathcal{J}|})$ (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.(4) 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 $\Delta\mathbf{o} = \{\Delta o_1, \cdots, \Delta o_{|\mathcal{J}|}\}$.

procedure DECREASE_WORKLOADS($\mathbf{o}$, g)
/* The procedure NEXT_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.(7).
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. 3. The algorithm to decrease the workload tuple for a given global gradient.

From Eq.(1), $s_i[\mathbf{o}\langle g - \Delta g \rangle]$ is given by

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

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

$$g_i[\mathbf{o}\langle g - \Delta g \rangle] \;=\; \frac{P'(s_i[\mathbf{o}\langle g - \Delta g \rangle])}{f_i'(o_i[\mathbf{o}\langle g \rangle] - \Delta o_i)} \;\;, \tag{5}$$

which is an explicit expression of $\Delta\mathbf{o}$. Therefore, we can compute $\Delta\mathbf{o}$ by solving Eq.(4) 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 $O_1, O_2, \cdots, O_n$:

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

where $g_0 = 0$ and $g_n = \nabla[\mathbf{u}]$. We call $g_1, g_2, \cdots, g_{n-1}$ *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

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

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 $\sum_{J_i \in \mathcal{J}_k[\mathbf{o}\langle g_l \rangle]} a_i[\mathbf{o}\langle g_l \rangle]$, respectively. For each $J_i \in \mathcal{J}_k$, its workload $o_i[\mathbf{o}\langle g \rangle]$ is set to be an explicit expression

of h_k (h_k represents the decrease in the reward per unit decrease in the workload):

$$o_i[\mathbf{o}\langle g\rangle] \;=\; \gamma_i(h_k) \quad \text{where } \gamma_i \text{ is the function defined by} \tag{6}$$

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

h_k satisfies the following equation that corresponds to Eq.(4):

$$\frac{P'(\sum_{J_i \in \mathcal{J}_k} \gamma_i(h_k) \;/\; \|\mathbf{I}_k\|)}{h_k} \;=\; g\ . \tag{7}$$

Because the left-hand side of Eq.(7) is a strictly decreasing function of h_k (from the concavity of f_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 $\mathbf{o}\langle g\rangle$. The following lemma captures the heart of the procedure DECREASE_WORKLOADS.

Lemma 3. *For $g_{l-1} < g \le g_l$, let $\mathbf{o}\langle g\rangle$ be defined by Eq.(6) and Eq.(7). Then,*

$$\frac{dE(\mathbf{A}[\mathbf{o}\langle g\rangle])}{dF(\mathbf{o}\langle g\rangle)} \;=\; g \quad \textit{for all} \quad g_{l-1} < g \le g_l\ .$$

Proof. Let $O_k(g)$ denote the sum of workloads of jobs in $\mathcal{J}_k$ under $\mathbf{o}\langle g\rangle$, i.e., $O_k(g) = \sum_{J_i \in \mathcal{J}_k} o_i[\mathbf{o}\langle g\rangle]$. Let $F_k(g)$ and $E_k(g)$ denote the sum of rewards and the energy consumption of jobs in $\mathcal{J}_k$ under $\mathbf{A}[\mathbf{o}\langle g\rangle]$, respectively, i.e.,

$$F_k(g) \;=\; \sum_{J_i \in \mathcal{J}_k} f_i(o_i[\mathbf{o}\langle g\rangle]) \quad \text{and} \quad E_k(g) \;=\; P\left(\frac{O_k(g)}{\|\mathbf{I}_k\|}\right) \cdot \|\mathbf{I}_k\|\ .$$

For a job $J_i \in \mathcal{J}_k$ such that $m_i < o_i[\mathbf{o}\langle g\rangle] < u_i$, we have from Eq.(6):

$$\begin{aligned} &m_i < \gamma_i(h_k) < u_i\ , \quad \text{which implies} \quad \gamma_i(h_k) = f_i'^{-1}(h_k) \quad \text{and then} \\ &f_i'(o_i[\mathbf{o}\langle g\rangle]) \;=\; f_i'(\gamma_i(h_k)) \;=\; f_i'(f_i'^{-1}(h_k)) \;=\; h_k\ . \end{aligned}$$

Therefore,

$$\begin{aligned} \frac{dF_k(g)}{dO_k(g)} \;&=\; \sum_{J_i \in \mathcal{J}_k} \frac{dF_k(g)}{do_i[\mathbf{o}\langle g\rangle]} \frac{do_i[\mathbf{o}\langle g\rangle]}{dO_k(g)} \;=\; \sum_{J_i \in \mathcal{J}_k \,\wedge\, m_i < o_i[\mathbf{o}\langle g\rangle] < u_i} f_i'(o_i[\mathbf{o}\langle g\rangle]) \cdot \frac{do_i[\mathbf{o}\langle g\rangle]}{dO_k(g)} \\ &= h_k \cdot \sum_{J_i \in \mathcal{J}_k \,\wedge\, m_i < o_i[\mathbf{o}\langle g\rangle] < u_i} \frac{do_i[\mathbf{o}\langle g\rangle]}{dO_k(g)} \;=\; h_k \cdot \frac{dO_k(g)}{dO_k(g)} \;=\; h_k\ . \end{aligned} \tag{8}$$

Furthermore,

$$\frac{dE_k(g)}{dO_k(g)} \;=\; \frac{1}{\|\mathbf{I}_k\|} \cdot P'\left(\frac{O_k(g)}{\|\mathbf{I}_k\|}\right) \cdot \|\mathbf{I}_k\| \;=\; P'\left(\frac{O_k(g)}{\|\mathbf{I}_k\|}\right)\ . \tag{9}$$

From Eq.(8) , Eq.(9) and Eq.(7), we have

$$\frac{dE_k(g)}{dF_k(g)} \;=\; \frac{dO_k(g)}{dF_k(g)} \cdot \frac{dE_k(g)}{dO_k(g)} \;=\; \frac{1}{h_k} \cdot P'\left(\frac{O_k(g)}{\|\mathbf{I}_k\|}\right) \;=\; g\ . \tag{10}$$

Finally, from Eq.(10), it follows that

$$\frac{dE(\mathbf{A}[\mathbf{o}\langle g\rangle])}{dF(\mathbf{o}\langle g\rangle)} = \sum_{k\geq 1}\frac{dE_k(g)}{dF(\mathbf{o}\langle g\rangle)} = \sum_{k\geq 1}\left(\sum_{k\geq 1}\frac{dE_k(g)}{dF_k(g)}\cdot\frac{dF_k(g)}{dF(\mathbf{o}\langle g\rangle)}\right)$$
$$= \sum_{k\geq 1}\left(\sum_{k\geq 1} g\cdot\frac{dF_k(g)}{dF(\mathbf{o}\langle g\rangle)}\right) = g\cdot\frac{dF(\mathbf{o}\langle g\rangle)}{dF(\mathbf{o}\langle g\rangle)} = g\ . \qquad \square$$

Based on the invariant described by Lemma 3, we prove in Section 3.3 that $\mathbf{o}\langle g\rangle$ always passes through the optimal solution, i.e., the optimal solution $\mathbf{o}_{\text{opt}}$ is given by $\mathbf{o}\langle g_{\text{opt}}\rangle$ 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 be 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

$$\lim_{\varepsilon\to 0}\sigma_{k_1}[\mathbf{o}\langle g+\varepsilon\rangle] = \lim_{\varepsilon\to 0}\sigma_{k_2}[\mathbf{o}\langle g+\varepsilon\rangle]\ , \quad \text{which implies}$$
$$\lim_{\varepsilon\to 0}\frac{\sum_{J_i\in\mathcal{J}_{k_1}[\mathbf{o}\langle g+\varepsilon\rangle]} o_i[\mathbf{o}\langle g+\varepsilon\rangle]}{\sum_{J_i\in\mathcal{J}_{k_1}[\mathbf{o}\langle g+\varepsilon\rangle]} a_i[\mathbf{o}\langle g+\varepsilon\rangle]} = \lim_{\varepsilon\to 0}\frac{\sum_{J_i\in\mathcal{J}_{k_2}[\mathbf{o}\langle g+\varepsilon\rangle]} o_i[\mathbf{o}\langle g+\varepsilon\rangle]}{\sum_{J_i\in\mathcal{J}_{k_2}[\mathbf{o}\langle g+\varepsilon\rangle]} a_i[\mathbf{o}\langle g+\varepsilon\rangle]}\ . \qquad (11)$$

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

$$\lim_{\varepsilon\to 0}\sigma_{k_1}[\mathbf{o}\langle g-\varepsilon\rangle] = \lim_{\varepsilon\to 0}\sigma_{k_2}[\mathbf{o}\langle g-\varepsilon\rangle]\ , \quad \text{which implies}$$
$$\exists\, r,d\ \in\ \{\min I_i'\ ,\ \max I_i'\mid J_i\in\mathcal{J}_k[\mathbf{o}\langle g+\varepsilon\rangle]\}\ ,$$
$$\lim_{\varepsilon\to 0}\frac{\sum_{J_i\in\mathcal{J}_{[r,d]}} o_i[\mathbf{o}\langle g+\varepsilon\rangle]}{\|\mathbf{I}_k[\mathbf{o}\langle g+\varepsilon\rangle]\cap[r,d]\|} = \lim_{\varepsilon\to 0}\frac{\sum_{J_i\in\mathcal{J}_k[\mathbf{o}\langle g+\varepsilon\rangle]-\mathcal{J}_{[r,d]}} o_i[\mathbf{o}\langle g+\varepsilon\rangle]}{\|\mathbf{I}_k[\mathbf{o}\langle g+\varepsilon\rangle]\| - \|\mathbf{I}_k[\mathbf{o}\langle g+\varepsilon\rangle]\cap[r,d]\|}\ . (12)$$

Provided that the separating gradients $g_n, g_{n-1},\cdots,g_l$ 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.(11) 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.(12) by $\mathbf{I}_k[\mathbf{o}\langle g_l\rangle]$ (Note that the latter is already known.)
(c) Remove lim operators from Eq.(11) and Eq.(12) and replace $g+\varepsilon$ by g .
(d) Return the largest g $(<g_l)$ that satisfies the simultaneous equations Eq.(6), Eq.(7) and Eq.(11), or the equations Eq.(6), Eq.(7) and Eq.(12).

The above procedure makes good use of the property that $\mathbf{o}\langle g\rangle \approx \mathbf{o}\langle g_l\rangle$ for all $g_{l-1} < g \leq g_l$ 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_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.

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

Proof. Let α_i $(1\leq i\leq|\mathcal{J}|)$ be the functions on $(0,\nabla[\mathbf{u}]]$ defined by

$$\alpha_i(g) \;=\; \begin{cases} 0 & o_i[\mathbf{o}\langle g\rangle] > m_i \;, \\ 2\cdot|\mathcal{J}| & o_i[\mathbf{o}\langle g\rangle] = m_i \;. \end{cases}$$

Furthermore, for i and j such that $(r_i,d_i)\cap(r_j,d_j)\neq\emptyset$, let $\beta_{i,j}$ $(1\leq i<j\leq|\mathcal{J}|)$ be the functions on $(0,\nabla[\mathbf{u}]]$ defined by

$$\beta_{i,j}(g) \;=\; \begin{cases} 0 & (s_i[\mathbf{u}] > s_j[\mathbf{u}] \;\wedge\; s_i[\mathbf{o}\langle g\rangle] > s_j[\mathbf{o}\langle g\rangle]) \\ & \quad \vee\, (s_i[\mathbf{u}] < s_j[\mathbf{u}] \;\wedge\; s_i[\mathbf{o}\langle g\rangle] < s_j[\mathbf{o}\langle g\rangle]) \\ & \quad \vee\, (s_i[\mathbf{u}] = s_j[\mathbf{u}] \;\wedge\; s_i[\mathbf{o}\langle g\rangle] = s_j[\mathbf{o}\langle g\rangle]) \\ 1 & (s_i[\mathbf{u}] > s_j[\mathbf{u}] \;\wedge\; s_i[\mathbf{o}\langle g\rangle] = s_j[\mathbf{o}\langle g\rangle]) \\ & \quad \vee\, (s_i[\mathbf{u}] < s_j[\mathbf{u}] \;\wedge\; s_i[\mathbf{o}\langle g\rangle] = s_j[\mathbf{o}\langle g\rangle]) \\ & \quad \vee\, (s_i[\mathbf{u}] = s_j[\mathbf{u}] \;\wedge\; s_i[\mathbf{o}\langle g\rangle] \neq s_j[\mathbf{o}\langle g\rangle]) \\ 2 & (s_i[\mathbf{u}] > s_j[\mathbf{u}] \;\wedge\; s_i[\mathbf{o}\langle g\rangle] < s_j[\mathbf{o}\langle g\rangle]) \\ & \quad \vee\, (s_i[\mathbf{u}] < s_j[\mathbf{u}] \;\wedge\; s_i[\mathbf{o}\langle g\rangle] > s_j[\mathbf{o}\langle g\rangle]) \;. \end{cases}$$

Finally, let $\delta(g) \;=\; \sum_{i=1}^{|\mathcal{J}|}\alpha_i(g) \;+\sum_{(r_i,d_i)\cap(r_j,d_j)\neq\emptyset}\beta_{i,j}(g)$. It can be easily verified that for each separating gradient g_l, $\delta(g_l+\varepsilon)+1\leq\delta(g_l-\varepsilon)$, i.e., δ strictly increases around g_l. However, because $\delta(\nabla[\mathbf{u}])=0$ and $\delta(g)$ is bounded by $4\cdot|\mathcal{J}|^2$, the number of separating gradient within $(0,\nabla[\mathbf{u}]]$ is bounded by $4\cdot|\mathcal{J}|^2$. □

3.3 Optimality Proof

In this section, we prove that the algorithm described in Section 3.2 always computes the optimal solution. Before presenting the optimality proof, we define additional notations and prove some miscellaneous properties.

In Section 3.2, $\mathbf{o}\langle g\rangle$ was used to search the solution space given by $\{\mathbf{o}\mid\mathbf{m}\leq\mathbf{o}\leq\mathbf{u}\}$. For the optimality proof, we consider a restricted solution space given by $\{\mathbf{o}\mid\mathbf{m}'\leq\mathbf{o}\leq\mathbf{u}\}$ where $\mathbf{m}\leq\mathbf{m}'$. Note that the lower bound is increased from $\mathbf{m}$ to $\mathbf{m}'$. $\mathbf{o}\langle g\rangle(\mathbf{m}')$ is used to denote g's corresponding workload tuple when the algorithm in Section 3.2 takes $\mathbf{m}'$ instead of $\mathbf{m}$ as input. From the definition, $\mathbf{o}\langle g\rangle(\mathbf{m})\equiv\mathbf{o}\langle g\rangle$ for all g.

Lemma 5. *Given* $\mathbf{m}'$ $(\mathbf{m}\leq\mathbf{m}')$ *and* $o_i[\mathbf{m}']<x<u_i$, *let* $g_i(x,\mathbf{m}')$ *denote the unique* g *that satisfies* $o_i[\mathbf{o}\langle g\rangle(\mathbf{m}')]=x$. *Then, for* $\mathbf{o}_1$ *and* $\mathbf{o}_2$ *such that* $\mathbf{m}\leq\mathbf{o}_1\leq\mathbf{o}_2\leq\mathbf{u}$ *and* $o_i[\mathbf{o}_2]<x_i<u_i$, *we have* $g_i(x_i,\mathbf{o}_1)\;\leq\;g_i(x_i,\mathbf{o}_2)$.

Proof. Let $\mathbf{o}_g$ and $\mathbf{o}'_g$ denote $\mathbf{o}\langle g_i(x_i,\mathbf{o}_1)\rangle(\mathbf{o}_1)$ and $\mathbf{o}\langle g_i(x_i,\mathbf{o}_2)\rangle(\mathbf{o}_2)$, respectively. Then, it suffices to show that $s_i[\mathbf{o}_g]\leq s_i[\mathbf{o}'_g]$ since $g_i(x_i,\mathbf{o}_1)=P'(s_i[\mathbf{o}_g])/f_i'(x_i)$ and $g_i(x_i,\mathbf{o}_2)=$

$P'(s_i[\mathbf{o}'_g])/f'_i(x_i)$. To find explicit representations of $s_i[\mathbf{o}_g]$ and $s_i[\mathbf{o}'_g]$, we first define $\mathbf{o}_3$ and $\mathbf{o}_4$ as follows:

$$o_j[\mathbf{o}_3] = \begin{cases} x_i & j = i \\ u_j & f'_j(u_j) > f'_i(x_i) \ , \\ o_j[\mathbf{o}_1] & f'_j(o_j[\mathbf{o}_1]) < f'_i(x_i) \ , \\ f'^{-1}_j(f'_i(x_i)) & \text{otherwise.} \end{cases}$$

$$o_j[\mathbf{o}_4] = \begin{cases} x_i & j = i \\ u_j & f'_j(u_j) > f'_i(x_i) \ , \\ o_j[\mathbf{o}_2] & f'_j(o_j[\mathbf{o}_2]) < f'_i(x_i) \ , \\ f'^{-1}_j(f'_i(x_i)) & \text{otherwise.} \end{cases}$$

The definition of $\mathbf{o}_3$ and $\mathbf{o}_4$ implies $\mathbf{o}_3 \leq \mathbf{o}_4$, since if $j \neq i$ and $f'_j(u_j) \leq f'_i(x_i)$ we have

$$o_j[\mathbf{o}_3] = \max\{o_j[\mathbf{o}_1], f'^{-1}_j(f'_i(x_i))\} \leq \max\{o_j[\mathbf{o}_2], f'^{-1}_j(f'_i(x_i))\} = o_j[\mathbf{o}_4] \ . \quad (13)$$

Since $o_i[\mathbf{o}_2] < x_i < u_i$, we have for some k and k':

$$J_i \in \mathcal{J}_k[\mathbf{o}_g] \ (\in \mathcal{G}[\mathbf{o}_g]) \ \text{ and } \ J_i \in \mathcal{J}_{k'}[\mathbf{o}_3] \ (\in \mathcal{G}[\mathbf{o}_3]) \ .$$

We would like to show that

$$\forall J_j \in \mathcal{J}_k[\mathbf{o}_g] \ , \ o_j[\mathbf{o}_g] = o_j[\mathbf{o}_3] \quad (14)$$
$$\forall J_j \in \cup_{h<k}\mathcal{J}_h[\mathbf{o}_g] \ , \ o_j[\mathbf{o}_g] \leq o_j[\mathbf{o}_3] \ , \text{ and} \quad (15)$$
$$\forall J_j \in \cup_{h>k}\mathcal{J}_h[\mathbf{o}_g] \ , \ o_j[\mathbf{o}_g] \geq o_j[\mathbf{o}_3] \ . \quad (16)$$

For $J_j \in \mathcal{J}_k[\mathbf{o}_g]$, $o_j[\mathbf{o}_g]$ is given by $o_j[\mathbf{o}_g] = \max\{o_j[\mathbf{o}_1], f'^{-1}_j(f'_i(x_i))\} = o_j[\mathbf{o}_3]$.
For the case that $J_j \in \mathcal{J}_h[\mathbf{o}_g]$ where $h < k$, we have

$$o_j[\mathbf{o}_g] = o_j[\mathbf{o}_1] \quad \text{or} \quad o_j[\mathbf{o}_g] \leq f'^{-1}_j(f'_i(x_i)) \quad (\text{ i.e., } \ f'_j(o_j[\mathbf{o}_g]) \geq f'_i(x_i) \ ) \ ,$$

since otherwise

$$g_i(x_i, \mathbf{o}_1) = \nabla_h[\mathbf{o}_g] = g_j[\mathbf{o}_g] = \frac{P'(s_j[\mathbf{o}_g])}{f'_j(o_j[\mathbf{o}_g])} > \frac{P'(s_i[\mathbf{o}_g])}{f'_i(x_i)} = g_i(x_i, \mathbf{o}_1) \ ,$$

a contradiction. Therefore, we have

$$o_j[\mathbf{o}_g] \leq \max\{o_j[\mathbf{o}_1], f'^{-1}_j(f'_i(x_i))\} \leq o_j[\mathbf{o}_3] \ .$$

Finally, for the case that $J_j \in \mathcal{J}_h[\mathbf{o}_g]$ where $h > k$, J_j must also be included in $\mathcal{J}_h^{\top}[\mathbf{o}_g]$.
Therefore,

$$f'_j(o_j[\mathbf{o}_g]) = \frac{P'(s_j[\mathbf{o}_g])}{g_i(x_i, \mathbf{o}_1)} = \frac{P'(s_j[\mathbf{o}_g])}{P'(s_i[\mathbf{o}_g])} \cdot f'_i(x_i) \leq f'_i(x_i) \ ,$$

which implies $o_j[\mathbf{o}_g] \geq f'^{-1}_j(f'_i(x_i))$. Furthermore, since we have $f'_j(u_j) \ (\leq f'_j(o_j[\mathbf{o}_g]) \leq f'_i(x_i)$, it follows that $o_j[\mathbf{o}_3] = \max\{o_j[\mathbf{o}_1], f'^{-1}_j(f'_i(x_i))\} \leq o_j[\mathbf{o}_g]$. Thus, (14)-(16) hold, which implies $s_i[\mathbf{o}_g] = \sigma_k[\mathbf{o}_g] = \sigma_{k'}[\mathbf{o}_3] = s_i[\mathbf{o}_3]$.
By applying the same argument to $\mathbf{o}'_g$ and $\mathbf{o}_4$, we have $s_i[\mathbf{o}'_g] = s_i[\mathbf{o}_4]$, and consequently, from (13) and Lemma 1, $s_i[\mathbf{o}_g] = s_i[\mathbf{o}_3] \leq s_i[\mathbf{o}_4] = s_i[\mathbf{o}'_g]$. □

Lemma 6. *For* $\mathbf{m} \leq \mathbf{m}'$ *and* $0 < g \leq \nabla[\mathbf{u}]$,

$$\forall\, 1 \leq i \leq |\mathcal{J}| \ , \ \ s_i[\mathbf{o}\langle g\rangle(\mathbf{m})] \ \leq \ s_i[\mathbf{o}\langle g\rangle(\mathbf{m}')] \ .$$

Proof. To begin with, we prove that

$$\begin{aligned} \forall\, 1 \leq i \leq |\mathcal{J}| \ , \ \forall\, g_i(o_i[\mathbf{m}'],\mathbf{m}') \ \leq \ g \ \leq \ g_i(u_i,\mathbf{m}') \ , \\ s_i[\mathbf{o}\langle g\rangle(\mathbf{m})] \ \leq \ s_i[\mathbf{o}\langle g\rangle(\mathbf{m}')] \ , \end{aligned} \tag{17}$$

which is a weakened version of the lemma. Suppose to the contrary that for some i and $g_i(o_i[\mathbf{m}'],\mathbf{m}') \ \leq \ g \ \leq \ g_i(u_i,\mathbf{m}')$, $s_i[\mathbf{o}\langle g\rangle(\mathbf{m})] \ > \ s_i[\mathbf{o}\langle g\rangle(\mathbf{m}')]$.

For the case that $g_i(m_i,\mathbf{m}) \ \leq \ g \ \leq \ g_i(u_i,\mathbf{m})$, J_i is included in both $\cup_{\nabla_k[\mathbf{o}\langle g\rangle(\mathbf{m})]=g}\, \mathcal{J}_k^{\top}[\mathbf{o}\langle g\rangle(\mathbf{m})]$ and $\cup_{\nabla_k[\mathbf{o}\langle g\rangle(\mathbf{m}')]=g}\, \mathcal{J}_k^{\top}[\mathbf{o}\langle g\rangle(\mathbf{m}')]$, that is,

$$\begin{aligned} & g_i[\mathbf{o}\langle g\rangle(\mathbf{m})] \ = \ g \ = \ g_i[\mathbf{o}\langle g\rangle(\mathbf{m}')] \ , \ \ \text{which implies} \\ & \frac{P'(s_i[\mathbf{o}\langle g\rangle(\mathbf{m})])}{f_i'(o_i[\mathbf{o}\langle g\rangle(\mathbf{m})])} \ = \ \frac{P'(s_i[\mathbf{o}\langle g\rangle(\mathbf{m}')])}{f_i'(o_i[\mathbf{o}\langle g\rangle(\mathbf{m}')])} \ . \end{aligned}$$

From the assumption, we have

$$o_i[\mathbf{o}\langle g\rangle(\mathbf{m})] \ < \ o_i[\mathbf{o}\langle g\rangle(\mathbf{m}')] \quad (\text{ i.e., } \ f_i'(o_i[\mathbf{o}\langle g\rangle(\mathbf{m})]) \ > \ f_i'(o_i[\mathbf{o}\langle g\rangle(\mathbf{m}')] \ ) \ ,$$

a contradiction. (Recall that $g_i(o_i[\mathbf{m}'],\mathbf{m}') \ \leq \ g \ \leq \ g_i(u_i,\mathbf{m}')$ and $g_i(m_i,\mathbf{m}) \ \leq \ g \ \leq \ g_i(u_i,\mathbf{m})$.)

For the other case, i.e., $g > g_i(u_i,\mathbf{m})$ (Note that $g_i(u_i,\mathbf{m}) \leq g_i(o_i[\mathbf{m}'],\mathbf{m}'])$.) , J_i is included in $\cup_{\nabla_k[\mathbf{o}\langle g\rangle(\mathbf{m})]=g}\, \mathcal{J}_k^{\top}[\mathbf{o}\langle g\rangle(\mathbf{m})]$, but not in $\cup_{\nabla_k[\mathbf{o}\langle g\rangle(\mathbf{m}')]=g}\, \mathcal{J}_k^{\top}[\mathbf{o}\langle g\rangle(\mathbf{m}')]$. Therefore,

$$\begin{aligned} & \frac{P'(s_i[\mathbf{o}\langle g\rangle(\mathbf{m})])}{f_i'(o_i[\mathbf{o}\langle g\rangle(\mathbf{m})])} \ < \ g \ = \ \frac{P'(s_i[\mathbf{o}\langle g\rangle(\mathbf{m}')])}{f_i'(o_i[\mathbf{o}\langle g\rangle(\mathbf{m}')])} \ , \ \ \text{which implies} \\ & u_i \ = \ o_i[\mathbf{o}\langle g\rangle(\mathbf{m})] \ < \ o_i[\mathbf{o}\langle g\rangle(\mathbf{m}')] \ \leq \ u_i \ , \end{aligned}$$

a contradiction. Thus, Eq.(17) is proved.

Based on Eq.(17), we extend the result to the case of arbitrary $0 < g \leq \nabla[\mathbf{u}]$. Suppose that for some i and $0 < g \leq \nabla[\mathbf{u}]$, $s_i[\mathbf{o}\langle g\rangle(\mathbf{m})] \ > \ s_i[\mathbf{o}\langle g\rangle(\mathbf{m}')]$. Then, we have either $g > g_i(u_i,\mathbf{m}')$ or $g < g_i(o_i[\mathbf{m}'],\mathbf{m}')$.

For the case that $g > g_i(u_i,\mathbf{m}')$, from Lemma 5, we have

$$g_i(u_i,\mathbf{m}') \ > \ g_i(u_i,\mathbf{m}) \quad \text{and} \quad o_i[\mathbf{o}\langle g\rangle(\mathbf{m}')] \ = \ o_i[\mathbf{o}\langle g\rangle(\mathbf{m})] \ = \ u_i$$

and, from Lemma 1, $s_i[\mathbf{o}\langle g\rangle(\mathbf{m}')] \ < \ s_i[\mathbf{o}\langle g\rangle(\mathbf{m})] \ \leq \ s_i[\mathbf{u}]$. Therefore, for some $g' \geq g$ and k such that $J_i \ \in \ \mathcal{J}_k[\mathbf{o}\langle g'\rangle(\mathbf{m}')] \ (\ \in \ \mathcal{G}[\mathbf{o}\langle g'\rangle(\mathbf{m}')] \ )$

$$J_i \ \in \ \mathcal{J}_k^{+}[\mathbf{o}\langle g'\rangle(\mathbf{m}')] \ - \ \mathcal{J}_k^{\top}[\mathbf{o}\langle g'\rangle(\mathbf{m}')] \quad , \quad \mathcal{J}_k^{\top}[\mathbf{o}\langle g'\rangle(\mathbf{m}')] \ \neq \ \emptyset \quad \text{and} \tag{18}$$

$$s_i[\mathbf{o}\langle g'\rangle(\mathbf{m}')] \ < \ s_i[\mathbf{o}\langle g'\rangle(\mathbf{m})] \ \leq \ s_i[\mathbf{u}] \ . \tag{19}$$

Note that $o_i[\mathbf{o}\langle g'\rangle(\mathbf{m}')] = o_i[\mathbf{o}\langle g'\rangle(\mathbf{m})] = u_i$. Let J_j be a job in $\mathcal{J}_k^{\top}[\mathbf{o}\langle g'\rangle(\mathbf{m}')]$. Since $g_j(o_j[\mathbf{o}'],\mathbf{m}') < g' < g_j(u_j,\mathbf{m}')$, we have from Eq.(17),

$$\begin{aligned} s_j[\mathbf{o}\langle g'\rangle(\mathbf{m})] \le s_j[\mathbf{o}\langle g'\rangle(\mathbf{m}')] &= \sigma_k[\mathbf{o}\langle g'\rangle(\mathbf{m}')] \\ &= s_i[\mathbf{o}\langle g'\rangle(\mathbf{m}')] < s_i[\mathbf{o}\langle g'\rangle(\mathbf{m})] \ , \qquad \text{(from Eq.(19).)} \end{aligned} \tag{20}$$

Since $J_i, J_j \in \mathcal{J}_k[\mathbf{o}\langle g'\rangle(\mathbf{m}')]$, Eq.(20) implies $u_i = o_i[\mathbf{o}\langle g'\rangle(\mathbf{m})] = 0$, a contradiction.

By applying the same argument to the case that $g < g_i(o_i[\mathbf{o}'],\mathbf{m}')$, we can prove that the assumption leads to a contradiction. Therefore, Eq.(17) also holds for an arbitrary $0 < g \le \nabla[\mathbf{u}]$. □

Lemma 7. *For* $\mathbf{m} \le \mathbf{m}'$ *and* $0 < e \le E(\mathcal{S}[\mathbf{u}])$ *, let* $g_1(e)$ *and* $g_2(e)$ *be defined by*

$$E(\mathcal{S}[\mathbf{o}\langle g_1(e)\rangle(\mathbf{m})]) = E(\mathcal{S}[\mathbf{o}\langle g_2(e)\rangle(\mathbf{m}')]) = e \ ,$$

respectively. Then,

$$F(\mathbf{o}\langle g_1(e)\rangle(\mathbf{m})) \ge F(\mathbf{o}\langle g_2(e)\rangle(\mathbf{m}')) \ .$$

Proof. For brevity, let $F_1(g) \stackrel{\text{def}}{=} F(\mathbf{o}\langle g\rangle(\mathbf{m}))$ and $F_2(g) \stackrel{\text{def}}{=} F(\mathbf{o}\langle g\rangle(\mathbf{m}'))$, respectively, and let $E_1(g) \stackrel{\text{def}}{=} E(\mathcal{S}[\mathbf{o}\langle g\rangle(\mathbf{m})])$ and $E_2(g) \stackrel{\text{def}}{=} E(\mathcal{S}[\mathbf{o}\langle g\rangle(\mathbf{m}')])$. Note that $F_1(g)$, $F_2(g)$, $E_1(g)$ and $E_2(g)$ are continuous and nondecreasing in g and $E_1(g_1(e)) = E_2(g_2(e)) = e$. Since E_1^{-1} is nondecreasing and $E_1(g_2(e)) \le E_2(g_2(e))$ by Lemmas 6 and 2, we have

$$g_2(e) = E_1^{-1}(E_1(g_2(e))) \le E_1^{-1}(E_2(g_2(e))) = E_1^{-1}(e) = g_1(e) \ . \tag{21}$$

From Lemma 3, we have

$$\frac{dF_1(g)}{dE_1(g)} = \frac{dF_2(g)}{dE_2(g)} = \frac{1}{g} \ .$$

It immediately follows that

$$\begin{aligned} F_1(g_1(e)) &= F(\mathbf{u}) - \int_e^{E(\mathcal{S}[\mathbf{u}])} \frac{1}{g_1(x)}\,dx \quad \text{and} \\ F_2(g_2(e)) &= F(\mathbf{u}) - \int_e^{E(\mathcal{S}[\mathbf{u}])} \frac{1}{g_2(x)}\,dx \ . \end{aligned}$$

From Eq.(21), we finally have $F_1(g_1(e)) \ge F_2(g_2(e))$. □

Now, we can prove the optimality of our algorithm.

Theorem 1. *The procedure* MAXIMIZE_REWARD *always returns the optimal solution.*

Proof. Note that the procedure MAXIMIZE_REWARD returns $\mathbf{o}\langle g_0\rangle(\mathbf{m})$ such that $E(\mathcal{S}[\mathbf{o}\langle g_0\rangle(\mathbf{m})]) = E_{\text{budget}}$. Suppose to the contrary that there exists $\mathbf{o}' \ne \mathbf{o}\langle g_0\rangle(\mathbf{m})$ such that

$$E(\mathcal{S}[\mathbf{o}']) = E_{\text{budget}} \ (= E(\mathcal{S}[\mathbf{o}\langle g_0\rangle(\mathbf{m})])\ ) \quad \text{and} \quad F(\mathbf{o}') > F(\mathbf{o}\langle g_0\rangle(\mathbf{m})) \ . \tag{22}$$

Since $\mathbf{o}' \equiv \mathbf{o}\langle\nabla[\mathbf{o}']\rangle(\mathbf{o}')$ and $\mathbf{m} \le \mathbf{o}'$, from Lemma 7, we have

$$F(\mathbf{o}') = F(\mathbf{o}\langle\nabla[\mathbf{o}']\rangle(\mathbf{o}')) \le F(\mathbf{o}\langle g_0\rangle(\mathbf{m})) \ ,$$

which contradicts Eq.(22). □

4 On-Line Algorithm

On-line reward-based voltage scheduling differs from 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 total rewards associated with optional workloads. Therefore, our on-line algorithm manages *energy slack* as well as slack time.

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 $E'(t) < E(t)$. 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 slack time because it can be directly detected and distributed among jobs executing next while for the slack time the preemption driven by the priority makes the analysis complicated.

With regard to slack time management, conventional voltage scheduling consists of two parts: slack time estimation and slack distribution. The goal of the slack estimation part is to identify as much slack time as possible while the goal of the slack distribution part is to distribute the slack time so that the resulting voltage schedule is as flat as possible. We adopt the existing slack time estimation method. However, in distributing the slack, we consider both the energy slack and the slack time, and try to increase the reward as much as possible by fully utilizing the energy slack as well as slack time.

In distributing two kinds of slacks, we exploits two properties that a maximum-reward voltage schedule exhibits. First, the voltage schedule (as a function of time) should be as flat as possible, as in an energy-optimal voltage schedule. Note that the optimality proof in Section 3.3 implies that the maximum-reward schedule for a given energy budget is also the energy-optimal schedule among those with the same reward. Second, the gradients of jobs, i.e., $P'(s_i)/f_i'$ should also be as flat as possible. Note that the optimal off-line algorithm tries to keep the gradient level as uniform as possible.

Assume that the slack time Δt and the energy slack ΔE are available at t and can be distributed among jobs $J_{i_1}, J_{i_2}, \cdots, J_{i_n}$ 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 $j = 1, 2, \cdots, n$ such that

$$\frac{P'(s_{i_j} + \Delta s_{i_j})}{f'_{i_j}((s_{i_j} + \Delta s_{i_j}) \cdot (a_{i_j} + \Delta a_{i_j}))} \tag{23}$$

is as uniform as possible subject to

$$\Delta t \geq \sum_{j=1}^{n} \Delta a_{i_j} \quad \text{and} \quad \Delta E \geq \sum_{j=1}^{n} P(s_{i_j} + \Delta s_{i_j}) \cdot (a_{i_j} + \Delta a_{i_j}) - P(s_{i_j}) \cdot a_{i_j} \ . \tag{24}$$

From the convexity of P and concavity of f, the gradient of J_{i_j} given by Eq.(23) increases both with Δa_{i_j} and with Δs_{i_j}. Thus, it is natural to assign higher Δa_{i_j} and Δs_{i_j} to a job with lower gradient. Our on-line algorithm first distributes Δt by increasing Δa_{i_j}'s iteratively until $\sum_{j=1}^{n} \Delta a_{i_j}$ reaches the available slack time Δt and then increases

Δs_{i_j}'s to distribute the remaining energy slack. Although this policy is very simple, it is sufficiently efficient as shown in the next section.

5 Experimental Results

In order to evaluate the performance of the proposed on-line algorithm in Section 4, we performed several experiments against the optimal off-line algorithm described in Section 3. For a comparison, the optimal off-line algorithm computes the theoretical optimal solution with the complete execution trace information. We used logarithmic functions of the type $\alpha_i \cdot \log(\beta_i \cdot o_i + 1)$ where the coefficients α_i and β_i were randomly chosen from uniform distributions within $[1, 10]$ and $[1, 50]$, respectively. We also performed experiments using linear reward functions of the type $\eta_i \cdot o_i$ where a uniform distribution within $[1, 10]$ was used for the coefficients η_i. We performed experiments using synthesized job sets with the varying number of jobs from 50 to 1600. In the experiments, the on-line algorithm was sufficiently efficient; the result obtained by the on-line algorithm is only $3 \sim 13\%$ worse than the theoretical optimal solution computed by the optimal off-line algorithm.

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. Experimental results show that the quality of solution computed by the on-line algorithm is only $3 \sim 13\%$ worse than that of the theoretically 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 a limited number of voltage levels and transition overheads in time and energy. In addition, we plan to develop off-line and on-line algorithms for fixed-priority real-time systems.

References

1. H. Aydin, 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. H. Aydin, 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. A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. SIAM, 2001.
4. J.-Y. Chung, 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. J.K. Dey, 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. F. Gruian. Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors. In *Proc. of International Symposium on Low Power Electronics and Design*, pages 46–51, 2001.
7. I. Hong, 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*, pages 178–187, 1998.
8. W. Kim, 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. W. Kim, 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. W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min. Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems. In *Proc. of Real-Time and Embedded Technology and Applications Symposium*, 2002.
11. W.-C. Kwon and T. Kim. Optimal Voltage Allocation Techniques for Dynamically Variable Voltage Processors. In *Proc. of Design Automation Conference*, pages 125–130, 2003.
12. K.-J. Lin, S. Natarajan, and J.W.-S. Liu. Imprecise Results: Utilizing Partial Computations in Real-Time Systems. In *Proc. of Real-Time Systems Symposium*, pages 210–217, 1987.
13. W.-S. Liu. *Real-Time Systems*. Prentice Hall, 2000.
14. P. Pillai 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. C. Rusu, R. Melhem, and D. Mossé. Maximizing the System Value While Satisfying Time and Energy Constraints. In *Proc. of Real-Time Systems Symposium*, pages 246–255, 2002.
16. C. Rusu, 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. T. Sakurai 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. W.-K. Shih, 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. Y. Shin and K. Choi. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In *Proc. of Design Automatioin Conference*, pages 134–139, 1999.
20. F. Yao, A. Demers, and S. Shenker. A Scheduling Model for Reduced CPU Energy. In *Proc. of IEEE Annual Foundations of Computer Science*, pages 374–382, 1995.
21. H.-S. Yun 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.