On Energy-Optimal Off-Line Scheduling for Fixed-Priority Hard Real-Time Systems On a Variable Speed Processor

Han-Saem Yun
School of Computer Science and Engineering
Seoul National University
Seoul, Korea 151-742
hsyun@davinci.snu.ac.kr

Abstract

Recently, there has been a wide spread of battery-operated embedded computing systems such as mobile and portable devices. For such systems, energy consumption is one of the most important design constraints because the battery operation time is a primary performance measure. Voltage scheduling, which adjusts the processor speed along with the supply voltage dynamically, is an effective technique in reducing the energy consumption of embedded real-time systems. Although many voltage scheduling algorithms have been proposed, there have been few research results known on the problem of energy-optimal off-line voltage scheduling for fixed-priority hard real-time systems. In this paper, we present three new contributions for this under-investigated problem. First, we prove that the problem is NP-hard. Second, we present a fully polynomial time approximation scheme (FPTAS) for the problem. For any $\varepsilon > 0$*, the proposed approximation scheme computes a voltage schedule whose energy consumption is at most* $(1+\varepsilon)$ *times that of the optimal voltage schedule. Furthermore, the running time is bounded by a polynomial function of the number of input jobs and* $1/\varepsilon$*. Third, we extend the FPTAS such that the average energy consumption is minimized given an on-line voltage scheduling algorithm and a probabilistic workload of a job set.*

1 Introduction

Embedded systems have emerged as one of the fastest growing areas of the computing world. This is most evident in the growth of battery-operated portable devices such as PDAs, mobile videophones, and cellular phones. For these devices, the most serious limitation is the available battery lifetime, and the energy consumption is a critical design constraint. Even for non-portable systems such as high-performance microprocessors, the energy consumption is still an important design constraint, because large heat dissipations in high-performance microprocessors may result in temperature-related problems such as logic errors or device degradation. As a consequence, several low-energy design techniques have been developed over a wide range of abstraction levels, including circuit, logic, architecture, compiler, OS, and application levels.

The dynamic energy consumption E, which dominates the total energy consumption of CMOS circuits, is given by $E \propto C_{\mathrm{L}} \cdot N_{\mathrm{cycle}} \cdot V_{\mathrm{DD}}^2$, where C_{L} is the load capacitance, N_{cycle} is the number of executed cycles, and V_{DD} is the supply voltage. Because the dynamic energy consumption E is quadratically dependent on the supply voltage V_{DD}, lowering V_{DD} is an effective technique in reducing the energy consumption. However, lowering the supply voltage also decreases the clock speed, because the circuit delay T_{D} of CMOS circuits is given by $T_{\mathrm{D}} \propto V_{\mathrm{DD}}/(V_{\mathrm{DD}} - V_{\mathrm{T}})^{\alpha}$ [15], where V_{T} is the threshold voltage and α is a technology-dependent constant.

When a given job does not require the maximum performance of a VLSI system, the clock speed (and its corresponding supply voltage) can be dynamically adjusted to the lowest possible level that still satisfies the job's required performance. This is the key principle of the voltage scheduling technique. With a recent explosive growth of the portable embedded system market, several commercial variable-speed processors were developed (e.g., Intel's *Xscale*, [20] AMD's *K6-2+* [19] and Transmeta's *Crusoe* [11] processors). Targeting these processors, various OS-level voltage scheduling algorithms [22, 5, 10, 1, 7, 17, 4, 18, 12, 13] have been proposed, especially for embedded hard real-time systems.

For hard real-time systems, the goal of voltage scheduling algorithms 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). In this paper, we consider *fixed-priority* real-time jobs running on a variable-speed processor.

1.1 Previous Work

Previous investigations on the voltage scheduling problem have focused mainly on real-time jobs running under dynamic-priority scheduling algorithms such as the EDF (earliest-deadline-first) algorithm [5, 10, 1, 7]. For example, the problem of energy-optimal EDF scheduling has been well understood. For EDF job sets, the algorithm by Yao *et al*. [22] computes the energy-optimal voltage schedules in polynomial time. Although the EDF scheduling policy makes the voltage scheduling problem easier to solve, fixed-priority scheduling algorithms such as the RM (rate monotonic) algorithm are more commonly used in practical real-time systems due to their low overhead and predictability [8].

Although there exist several voltage scheduling algorithms proposed for fixed-priority real-time tasks (e.g., on-line scheduling algorithms [17, 4, 10] and off-line scheduling algorithms [18, 4, 12, 13]), there have been few research results on the *optimal* voltage scheduling problem for fixed-priority hard real-time systems; neither a polynomial-time optimal voltage scheduling algorithm nor the computational complexity of the problem is known.

Up to now, the only significant research result on the optimality issue of fixed-priority voltage scheduling is the one presented by Quan *et al*. [13], where energy-optimal voltage schedules for fixed-priority jobs are found by an *exhaustive* algorithm. However, Quan *et al*. did not justify their exhaustive approach. If they had presented the computational complexity of the voltage scheduling problem, their result would have been much more significant. Since the worst-case complexity of Quan's algorithm is of higher order than $O(N!)$ where N is the number of jobs, the algorithm is practically unusable for most real-time applications.

Quan *et al*. also proposed a polynomial-time voltage scheduling algorithm for fixed-priority hard real-time systems [12], which is the best known polynomial-time heuristic for the problem. Although efficient, being a heuristic, this algorithm cannot guarantee the quality of the voltage schedule computed.

1.2 Contributions

In this paper, we give a complete treatment on the energy-optimal voltage scheduling problem for fixed-priority hard real-time systems. First, as with the work of Quan *et al*. [13, 12], we consider the optimal scheduling problem where the workload of each job is assumed to be constant, which we call the *Constant-Workload Optimal Scheduling* (CWOS) problem. The CWOS problem is identical to the one solved by Yao *et al.* [22] except that the priority assignment is changed from the dynamic EDF assignment to the fixed assignment. As illustrated by Quan *et al*. [12], the voltage scheduling problem for fixed-priority tasks is much more difficult to solve because the preemption relationship among the tasks is more complex to analyze.

We first prove that the CWOS problem is NP-hard, which implies that no optimal polynomial-time algorithm is likely to exist. Then, we present a *fully polynomial time approximation scheme* for the CWOS problem. A fully polynomial time approximation scheme (FPTAS) is an approximation algorithm that takes any ε (> 0) as an additional input and returns a solution whose cost is at most a factor of $(1+\varepsilon)$ away from the cost of the optimal solution with the running time bounded by a polynomial both in the size of the input instance and in $1/\varepsilon$ [21]. Given the NP-hardness of the problem, the proposed approximation scheme is practically the best solution. The proposed approximation scheme computes a near-optimal voltage schedule in polynomial time. By changing ε, the approximation scheme can find a voltage schedule that is provably arbitrarily close to the optimal solution.

Next, we consider the case where the workloads of jobs are not constant, which is a typical characteristic of real-world applications. For this case, on-line scheduling is necessary at runtime to exploit workload variations. Thus, we address the problem of computing an off-line voltage schedule that consumes the minimum *average* energy given an on-line voltage scheduling algorithm and a probabilistic workload of a job set, which we call the *Workload-Aware Optimal Scheduling* (WAOS) problem. Based on the average-case analysis of on-line voltage scheduling algorithms, we reduce the WAOS problem to the CWOS problem so that the FPTAS for the CWOS problem can be used for the WAOS problem with slight modification. As part of the analysis of on-line algorithms, we propose a generic on-line voltage scheduling algorithm, which is derived from the necessary conditions that any on-line algorithms should satisfy. The proposed generic on-line algorithm has its own significance in that it provides a new framework for the research on the on-line voltage scheduling. Experimental results show that our algorithm outperforms other well-known algorithms in the literature by up to about 40%.

The rest of the paper is organized as follows. In Section 2, we formulate the CWOS problem and characterize feasible voltage schedules. We describe important properties of an energy-optimal voltage schedule in Section 3, which provide a basis of later proofs. In Section 4, we present the intractability result for the problem including its NP-hardness. The FPTAS for the CWOS problem is presented in Section 5. In Section 6, we formulate the WAOS

problem, and present the generic on-line voltage scheduling algorithm and the FPTAS for the WAOS problem. Experimental results are given in Section 7 and we conclude with a summary and directions for future work in Section 8.

2 Problem Formulation

Note that only the CWOS problem is formulated in this section and the term "problem" indicates the CWOS problem up to Section 5; the formulation of the WAOS problem is given in Section 6.

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 $J \in \mathcal{J}$ is associated with the following timing parameters, which are assumed to be known off-line:

- r_J: the release time of J.
- d_J: the deadline of J.
- c_J: the number of execution cycles required for J.

We use p_J to denote the priority of the job J. We assume that J has a higher priority than J' if $p_J < p_{J'}$. In the rest of the paper, we use i instead of J_i as a subscript of timing parameters when no confusion arises. (e.g., r_i, d_i and c_i stand for r_{J_i}, d_{J_i} and c_{J_i}.) Note that our job model can be directly applicable to a periodic real-time system by considering all the task instances within a hyperperiod of periodic tasks.

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 in the rest of the paper. Given a voltage schedule, the job executed at time t can be uniquely determined and is denoted by $job(\mathcal{J}, \mathcal{S}, t)$. A voltage schedule $\mathcal{S}(t)$ is said to be *feasible* if $\mathcal{S}(t)$ gives each job the required number of cycles between its release time and deadline. (An exact characterization of a feasible voltage schedule is given in Section 2.1.)

As with other related work [22, 12, 13], we assume that the processor speed can be varied continuously with a negligible overhead both in time and power. Furthermore, we model that the power P, 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 is dependent only on the processor speed.

The goal of the voltage scheduling problem is, therefore, to find a feasible schedule $\mathcal{S}(t)$ that minimizes

$$E(\mathcal{S}) = \int_{t_s}^{t_f} P(\mathcal{S}(t))\, dt \tag{1}$$

where t_s and t_f are the lower and upper limits of release times and deadlines of the jobs in $\mathcal{J}$, respectively. For the rest of this paper, the energy-optimal voltage schedule of a job set $\mathcal{J}$ is denoted by $\mathcal{S}^{\mathcal{J}}_{\text{opt}}$.

2.1 Feasibility Analysis

In this section, we derive a necessary and sufficient condition for a voltage schedule to be feasible, which will provide a basis for the proofs in Section 3. We first introduce some useful notations and definitions.

$W(\mathcal{S}, [t_1, t_2])$ is used to denote the number of cycles executed under a voltage schedule $\mathcal{S}(t)$ from t_1 to t_2, i.e., $W(\mathcal{S}, [t_1, t_2]) = \int_{t_1}^{t_2} \mathcal{S}(t)\, dt$. Among $W(\mathcal{S}, [t_1, t_2])$ cycles, $W_i(\mathcal{S}, [t_1, t_2])$ denotes the number of cycles between t_1 and t_2 used for executing a set of jobs $J_1, J_2, \cdots, J_i$ whose priorities are higher than or equal to p_{J_i}. $R_{\mathcal{J}}$ and $D_{\mathcal{J}}$ represent the sets of release times and deadlines of the jobs in $\mathcal{J}$, respectively, i.e., $R_{\mathcal{J}} = \{r_J | J \in \mathcal{J}\}$ and $D_{\mathcal{J}} = \{d_J | J \in \mathcal{J}\}$. $T_{\mathcal{J}}$ denotes the union of $R_{\mathcal{J}}$ and $D_{\mathcal{J}}$, i.e., $T_{\mathcal{J}} = R_{\mathcal{J}} \cup D_{\mathcal{J}}$. Given a job set $\mathcal{J}' \subseteq \mathcal{J}$, $C(\mathcal{J}')$ represents the total workload of jobs in $\mathcal{J}'$, i.e., $C(\mathcal{J}') = \sum_{J \in \mathcal{J}'} c_J$. Furthermore, $\mathbf{I}_{\mathcal{J}'}$ represents the minimum interval that includes the execution intervals of jobs in $\mathcal{J}'$, i.e., $\mathbf{I}_{\mathcal{J}'} = [\min R_{\mathcal{J}'}, \max D_{\mathcal{J}'}]$. $\mathcal{T}^{\mathcal{J}}$ represents the cartesian product of $[r_{J_i}, d_{J_i}]$'s, for $1 \le i \le |\mathcal{J}|$, i.e., $\mathcal{T}^{\mathcal{J}} = [r_{J_1}, d_{J_1}] \times [r_{J_2}, d_{J_2}] \times \cdots \times [r_{J_{|\mathcal{J}|}}, d_{J_{|\mathcal{J}|}}]$. Given voltage schedules $\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_n$ such that

$$\mathcal{S}_i(t) = 0 \text{ for all } t \notin [\alpha_i, \beta_i] \text{ for all } 1 \le i \le n \quad \text{and} \quad \beta_i \le \alpha_{i+1} \text{ for all } 1 \le i < n \ ,$$

the concatenation of $\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_n$ is $\oplus_{i=1}^{n} \mathcal{S}_i = \mathcal{S}_1 \oplus \mathcal{S}_2 \oplus \cdots \oplus \mathcal{S}_n \stackrel{def}{=} \sum_{i=1}^{n} \mathcal{S}_i(t)$. Since jobs should be released before they can be processed, we assume that a voltage schedule $\mathcal{S}$ always satisfies the constraint that for any $t > 0$, $W(\mathcal{S}, [0, t]) \le C(\{J | r_J < t\})$.

The condition for a voltage schedule $\mathcal{S}(t)$ to be feasible can be expressed as follows:

Condition I (Feasibility Condition).

There exists a $|\mathcal{J}|$-tuple $(f_{J_1}, f_{J_2}, \cdots, f_{J_{|\mathcal{J}|}}) \in \mathcal{T}^{\mathcal{J}}$ such that
$\forall 1 \le i \le |\mathcal{J}| \quad \forall r \in \{t | t \in R_{\mathcal{J}} \wedge t < f_{J_i}\}$

$$W(\mathcal{S}, [r, f_{J_i}]) \ \ge \ C(\{J \,|p_J \le p_{J_i} \wedge r_J \in [r, f_{J_i})\}) \ . \quad (2)$$

For a $|\mathcal{J}|$-tuple $(f_{J_1}, f_{J_2}, \cdots, f_{J_{|\mathcal{J}|}}) \in \mathcal{T}^{\mathcal{J}}$, f_{J_i} can be considered as a modified deadline of J_i, which is equal to or precedes the original deadline d_{J_i}. (The meaning of the $|\mathcal{J}|$-tuple is further clarified in Section 3.) If $\mathcal{S}(t)$ satisfies Condition I for a given $|\mathcal{J}|$-tuple $(f_{J_1}, f_{J_2}, \cdots, f_{J_{|\mathcal{J}|}}) \in \mathcal{T}^{\mathcal{J}}$, J_i completes its execution by f_{J_i} for all $1 \le i \le |\mathcal{J}|$. Such $|\mathcal{J}|$-tuples are said to be *valid* with respect to $\langle \mathcal{J}, \mathcal{S}(t) \rangle$. Theorem 1 gives a proof for the feasibility condition.

Theorem 1 *Condition I is a necessary and sufficient condition for $\mathcal{S}(t)$ to be feasible.*

Proof. For the necessary part, suppose that $\mathcal{S}(t)$ is feasible, i.e., J_i completes its execution at $f_{J_i} \in (r_{J_i}, d_{J_i}]$ for all $1 \le i \le |\mathcal{J}|$. Then, for any $r \in R_{\mathcal{J}}$ such that $r < f_{J_i}$, all the higher priority jobs whose release times are within $[r, f_{J_i})$ complete their executions by f_{J_i}. So, the total amount of work that should be done within $[r, f_{J_i}]$ must be greater than or equal to the sum of workload of the jobs. Thus, we have for all $1 \le i \le |\mathcal{J}|$:

$$W(\mathcal{S}, [r, f_{J_i}]) \ \ge \ C(\{J \,|p_J \le p_{J_i} \ \wedge r_J \ \in [r, f_{J_i})\}) \ .$$

For the sufficient part, assume that Condition I is satisfied for a $|\mathcal{J}|$-tuple $(f_{J_1}, f_{J_2}, \cdots, f_{J_{|\mathcal{J}|}})$. By induction on i, we prove that J_i is given its required execution cycles c_{J_i} within $[r_{J_i}, f_{J_i}]$ for all $1 \le i \le |\mathcal{J}|$. The base case holds trivially.

For the induction step, assume that the proposition holds for all $k = 1, 2, \cdots, i-1$. Let $r < r_{J_i}$ be the earliest time point in $R_{\mathcal{J}}$ such that no lower priority jobs (i.e., J_k's for $k > i$) are executed within $[r, r_{J_i}]$, i.e., $W(\mathcal{S}, [r, r_{J_i}]) = W_{i-1}(\mathcal{S}, [r, r_{J_i}])$. If such r does not exist, r is set to r_{J_i}. Then, a higher priority job J' (i.e., J_l's for $l < i$) released before r (i.e., $r_{J'} < r$) must complete its execution before r; otherwise, since any lower priority jobs cannot be executed within $[r_{J'}, r]$, we have

$$W(\mathcal{S}, [r_{J'}, r_{J_i}]) \ = \ W(\mathcal{S}, [r_{J'}, r]) + W(\mathcal{S}, [r, r_{J_i}]) \ = \ W_{i-1}(\mathcal{S}, [r_{J'}, r]) + W_{i-1}(\mathcal{S}, [r, r_{J_i}]) \ = \ W_{i-1}(\mathcal{S}, [r_{J'}, r_{J_i}]) \ ,$$

which contradicts the definition of r. Since only higher priority jobs (i.e., J_l's for $l < i$) are executed within $[r, r_{J_i}]$, the amount of remaining workload of the higher priority jobs (which are released within $[r, r_{J_i})$) at time r_{J_i} is $C(\{J_k | 1 \le k < i \ \wedge \ r_{J_k} \in [r, r_{J_i})\}) \ - \ W(\mathcal{S}, [r, r_{J_i}])$. So, we have

$$\begin{aligned} W_{i-1}(\mathcal{S}, [r_{J_i}, f_{J_i}]) \ &\le \ C(\ \{J_k | 1 \le k < i \ \wedge \ r_{J_k} \in [r, r_{J_i})\}\ ) \ - \ W(\mathcal{S}, [r, r_{J_i}]) \ + \ C(\ \{J_k | 1 \le k < i \ \wedge \ r_{J_k} \in [r_{J_i}, f_{J_i})\}\ ) \\ &= \ C(\ \{J_k | 1 \le k < i \ \wedge \ r_{J_k} \in [r, f_{J_i})\}\ ) - \ W(\mathcal{S}, [r, r_{J_i}]) \ . \end{aligned} \quad (3)$$

To complete the induction, we only need to show that $W(\mathcal{S}, [r_{J_i}, f_{J_i}]) - W_{i-1}(\mathcal{S}, [r_{J_i}, f_{J_i}])$ is not smaller than c_{J_i}. (Note that J_i preempts any lower priority jobs.) From (3) and the assumption that Condition I is satisfied, we have

$$\begin{aligned} &W(\mathcal{S}, [r_{J_i}, f_{J_i}]) - W_{i-1}(\mathcal{S}, [r_{J_i}, f_{J_i}]) \\ &\quad \ge \ W(\mathcal{S}, [r, f_{J_i}]) \ - \ C(\ \{J_k | 1 \le k < i \ \wedge \ r_{J_k} \in [r, f_{J_i})\}\ ) && \text{(From (3).)} \\ &\quad \ge \ C(\{J_k \,| 1 \le k \le i \wedge r_{J_k} \in [r, f_{J_i})\}) \ - \ C(\ \{J_k | 1 \le k < i \ \wedge \ r_{J_k} \in [r, f_{J_i})\}\ ) && \text{(From (2).)} \\ &\quad = \ C(\{J_i\}) = c_{J_i} \ . \end{aligned}$$

□

A job set $\mathcal{J}$ is said to be an EDF job set if for any $J, J' \in \mathcal{J}$ (where $p_J < p_{J'}$), $d_J \le d_{J'}$ or $d_{J'} \le r_J$. When the priority assignment follows the EDF policy, we can prove that Condition I is simplified as follows:

Condition II (EDF Feasibility Condition).

For any $r \in R_{\mathcal{J}}$ and $d \in D_{\mathcal{J}}$ (where $r < d$),

$$W(\mathcal{S}, [r, d]) \ \ge \ C(\{J | [r_J, d_J] \subseteq [r, d]\}) \ .$$

Lemma 2 *Given an EDF job set $\mathcal{J}$, a voltage schedule $\mathcal{S}(t)$ of $\mathcal{J}$ is feasible if and only if Condition II is satisfied.*

Proof. Consider a new job set $\mathcal{J}' = \{J'_1, J'_2, \cdots, J'_{|\mathcal{J}|}\}$ where $r_{J'_i} = W(\mathcal{S}, [0, r_{J_i}])$, $d_{J'_i} = W(\mathcal{S}, [0, d_{J_i}])$, $c_{J'_i} = c_{J_i}$ and $p_{J'_i} = p_{J_i}$ for all $1 \le i \le |\mathcal{J}|$. Because $W(\mathcal{S}, [0,t])$ is a monotonically increasing function of t, $\mathcal{J}'$ is also an EDF job set (i.e., for any $J'_i, J'_k \in \mathcal{J}'$ where $i < k$, $d_{J'_i} \le d_{J'_k}$ or $d_{J'_k} \le r_{J'_i}$). Let $\mathcal{S}'(t) = 1$ $(\forall t > 0)$ be the voltage schedule of $\mathcal{J}'$. Then, we can easily verify that the index of the job $job(\mathcal{J}, \mathcal{S}, t)$ is the same as that of $job(\mathcal{J}', \mathcal{S}', W(\mathcal{S}, [0,t]))$. Therefore, $J_i \in \mathcal{J}$ finishes its execution by its deadline d_{J_i} under $\mathcal{S}(t)$ if and only if its corresponding job $J'_i \in \mathcal{J}'$ finishes its execution by $d_{J'_i}$ $(= W(\mathcal{S}, [0, d_{J_i}]))$ under $\mathcal{S}'$.

It is well known that all the jobs in an EDF job set meet their deadlines under a constant speed if and only if the utilization ratio for any time interval is less than or equal to 1 [8]. That is, $\mathcal{S}'$ is a feasible voltage schedule of $\mathcal{J}'$ if and only if the following is satisfied:

$$\text{For any } r' \in R_{\mathcal{J}'} \text{ and } d' \in D_{\mathcal{J}'} \text{ (where } r' < d'), \quad C(\{J | J \in \mathcal{J}' \wedge [r_J, d_J] \subseteq [r', d']\}) \le d' - r' . \tag{4}$$

Since (4) is equivalent to Condition II, Condition II is a necessary and sufficient condition for $\mathcal{S}(t)$ to be a feasible voltage schedule of $\mathcal{J}$. □

As shown in Conditions I and II, the complexity of fixed-priority voltage scheduling mainly comes from the inherent exhaustiveness in finding a valid $|\mathcal{J}|$-tuple. In the EDF scheduling algorithm, it is sufficient for a single $|\mathcal{J}|$-tuple of the original deadlines to be checked if it satisfies Condition II.

3 Some Properties of Optimal Schedules

In this section, we explain several properties for a feasible voltage schedule to be an energy-optimal schedule. These properties provide a key insight in devising a fast approximation algorithm described in Section 5. The first property, which was proven by Quan *et al.* [12] is that an energy-optimal voltage schedule should be a piecewise-constant function.

The existing optimal voltage scheduling algorithm by Quan *et al.* is based on an observation that if a given job set satisfies the requirement of an EDF job set, the optimal voltage schedule can be easily computed by Yao's "peak-power greedy" algorithm [22]. Simply applying Yao's algorithm to a fixed-priority job set may cause some jobs to miss their deadlines. However, if the deadlines of the jobs are appropriately modified before scheduling, Yao's algorithm can yield a feasible optimal schedule as shown in [13]. The efficiency of an optimal voltage scheduling algorithm is, therefore, dependent on how efficiently the job set is modified to be an EDF job set. To give a better insight into our approach for solving the voltage scheduling problem, we derive an equivalent result to Quan *et al.* [13] using Conditions I and II.

3.1 Properties on $|\mathcal{J}|$-Tuples

Given a $|\mathcal{J}|$-tuple $\mathbf{f} = (f_{J_1}, f_{J_2}, \cdots, f_{J_{|\mathcal{J}|}}) \in \mathcal{T}^{\mathcal{J}}$, $\mathcal{J}^{\mathbf{f}}$ represents the job set $\{J'_1, J'_2, \cdots, J'_{|\mathcal{J}|}\}$ where $p_{J'_i} = p_{J_i}, c_{J'_i} = c_{J_i}, r_{J'_i} = r_{J_i}$ and $d_{J'_i} = f_{J_i}$ for all $1 \le i \le |\mathcal{J}|$. We say that a $|\mathcal{J}|$-tuple $\mathbf{f}$ is *EDF-ordered* if $\mathcal{J}^{\mathbf{f}}$ follows the EDF priority. Furthermore, $\mathcal{J}^{\mathbf{f}}$ is said to be *EDF-equivalent* to $\mathcal{J}$. We first establish a link between Conditions I and II.

Lemma 3 *If Condition I is satisfied for a job set $\mathcal{J}$ by a voltage schedule $\mathcal{S}$ and an EDF-ordered $|\mathcal{J}|$-tuple $\mathbf{f} = (f_{J_1}, f_{J_2}, \cdots, f_{J_{|\mathcal{J}|}})$, Condition II is satisfied for a job set $\mathcal{J}^{\mathbf{f}}$ by $\mathcal{S}$.*

Proof. For any $r \in R_{\mathcal{J}^{\mathbf{f}}}$ and $d \in D_{\mathcal{J}^{\mathbf{f}}}$ $(r < d)$, we have

$$r \in \{t \mid t \in R_{\mathcal{J}} \ (= R_{\mathcal{J}^{\mathbf{f}}}) \wedge t < d\} \quad \text{and} \quad d = f_{J_i} \text{ for } \exists\, f_{J_i} \in D_{\mathcal{J}^{\mathbf{f}}} \ (= \{f_{J_1}, f_{J_2}, \cdots, f_{J_{|\mathcal{J}|}}\}) \ .$$

Furthermore, since $\mathbf{f}$ is EDF-ordered, we have

$$\forall J'_k \in \mathcal{J}^{\mathbf{f}} \text{ s.t. } r_{J'_k} \ (= r_{J_k}) \in [r, d(= f_{J_i})), \qquad \begin{array}{ll} d_{J'_k} = f_{J_k} \le f_{J_i} = d & \text{if } p_{J'_k} \le p_{J'_i} \ (= p_{J_i}) \\ d_{J'_k} = f_{J_k} > f_{J_i} = d & \text{otherwise.} \end{array}$$

Thus, we have for all $J'_k \in \mathcal{J}^{\mathbf{f}}$:

$$p_{J'_k} \le p_{J'_i} \wedge r_{J'_k} \in [r, d) \iff [r_{J'_k}, d_{J'_k}] \subseteq [r, d] . \tag{5}$$

Finally, by substituting d for f_{J_i} in (2), we have

$$\begin{aligned} W(\mathcal{S},[r,d]) &\geq C(\{J \in \mathcal{J} \mid p_J \leq p_{J_i} \wedge r_J \in [r,d)\}) = C(\{J'_k \in \mathcal{J}^{\mathbf{f}} \mid p_{J'_k} \leq p_{J'_i} (= p_{J_i}) \wedge r_{J'_k} (= r_{J_k}) \in [r,d)\}) \\ &= C(\{J' \in \mathcal{J}^{\mathbf{f}} \mid [r_{J'}, d_{J'}] \subseteq [r,d]\}) \ . \qquad \text{(From (5).)} \end{aligned}$$

□

Lemma 4 *If Condition II is satisfied for a job set $\mathcal{J}^{\mathbf{f}}$ by a voltage schedule $\mathcal{S}$ where $\mathbf{f} = (f_{J_1}, f_{J_2}, \cdots, f_{J_{|\mathcal{J}|}})$ is an EDF-ordered $|\mathcal{J}|$-tuple, Condition I is satisfied for a job set $\mathcal{J}$ by $\mathcal{S}$.*

Proof. Let $r \in \{t \mid t \in R_{\mathcal{J}} \wedge t < f_{J_i}\}$. Then, we have

$$r \in R_{\mathcal{J}^{\mathbf{f}}} (= R_{\mathcal{J}}) \ , \ f_{J_i} \in D_{\mathcal{J}^{\mathbf{f}}} (= \{f_{J_1}, f_{J_2}, \cdots, f_{J_{|\mathcal{J}|}}\}) \text{ and } r < f_{J_i}$$

and substituting f_{J_i} for d in Condition II gives

$$W(\mathcal{S},[r,f_{J_i}]) \geq C(\{J' \in \mathcal{J}^{\mathbf{f}} \mid [r_{J'}, d_{J'}] \subseteq [r, f_{J_i}]\}) \ .$$

Since $\mathbf{f}$ is EDF-ordered, we have for all $J'_k \in \mathcal{J}^{\mathbf{f}}$ (Refer to the proof of Lemma 3.):

$$p_{J'_k} \leq p_{J'_i} \wedge r_{J'_k} \in [r, f_{J_i}) \iff [r_{J'_k}, d_{J'_k}] \subseteq [r, f_{J_i}] \ . \tag{6}$$

Therefore, we have

$$\begin{aligned} W(\mathcal{S},[r,f_{J_i}]) &\geq C(\{J' \in \mathcal{J}^{\mathbf{f}} \mid [r_{J'}, d_{J'}] \subseteq [r, f_{J_i}]\}) = C(\{J'_k \in \mathcal{J}^{\mathbf{f}} \mid p_{J'_k} \leq p_{J'_i} (= p_{J_i}) \wedge r_{J'_k} (= r_{J_k}) \in [r, f_{J_i})\}) \\ &= C(\{J \in \mathcal{J} \mid p_J \leq p_{J_i} \wedge r_J \in [r, f_{J_i})\}) \ . \end{aligned}$$

□

From Lemmas 3 and 4, we can derive the following useful theorem which states how a feasible voltage schedule of a job set can be obtained from its EDF-equivalent job sets.

Theorem 5 *Given a job set $\mathcal{J}$, let $\mathcal{F}_{\mathcal{J}}$ be the set of all feasible voltage schedules for $\mathcal{J}$. Then, $\mathcal{F}_{\mathcal{J}} = \cup_{\mathbf{f} \in \mathcal{T}_{\text{EDF}}} \mathcal{F}_{\mathcal{J}^{\mathbf{f}}}$ where $\mathcal{T}_{\text{EDF}}$ is the set of all EDF-ordered $|\mathcal{J}|$-tuples for $\mathcal{J}$.*

Proof. To show that $\mathcal{S} \in \mathcal{F}_{\mathcal{J}} \Rightarrow \mathcal{S} \in \cup_{\mathbf{f} \in \mathcal{T}_{\text{EDF}}} \mathcal{F}_{\mathcal{J}^{\mathbf{f}}}$, assume that J_i completes its execution at $f_{J_i} (\leq d_{J_i})$ for all $1 \leq i \leq |\mathcal{J}|$ under $\mathcal{S} \in \mathcal{F}_{\mathcal{J}}$. Let $\mathbf{f} = (f_{J_1}, f_{J_2}, \cdots, f_{J_{|\mathcal{J}|}})$. Then, $\mathcal{J}^{\mathbf{f}}$ is an EDF job set. If not, we have for some $J'_k, J'_l \in \mathcal{J}^{\mathbf{f}}$ (where $p_{J'_k} < p_{J'_l}$)

$$r_{J'_k} < d_{J'_l} (= f_{J_l}) < d_{J'_k} (= f_{J_k}) \ ,$$

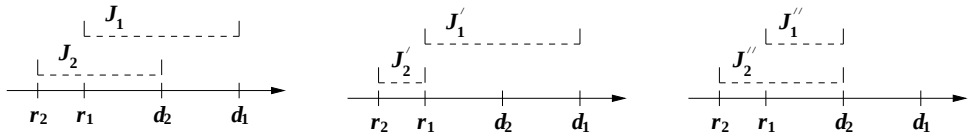
which contradicts a fact that once a higher priority job (i.e., J_k) is released during the execution of a lower priority job (i.e., J_l), the higher priority job completes earlier than the lower priority job (i.e., $f_{J_k} < f_{J_l}$). Furthermore, from Lemma 3, $\mathcal{S}(t)$ is a feasible schedule for the EDF job set $\mathcal{J}^{\mathbf{f}}$. Thus, we have $\mathcal{S} \in \cup_{\mathbf{f} \in \mathcal{T}_{\text{EDF}}} \mathcal{F}_{\mathcal{J}^{\mathbf{f}}}$.

Conversely, given an EDF-ordered $|\mathcal{J}|$-tuple $\mathbf{f} = (f_{J_1}, f_{J_2}, \cdots, f_{J_{|\mathcal{J}|}})$, let $\mathcal{S} \in \mathcal{F}_{\mathcal{J}^{\mathbf{f}}}$ be a feasible schedule for the EDF-equivalent job set $\mathcal{J}^{\mathbf{f}}$. Then, from Lemma 4, $\mathcal{S}$ satisfies Condition I for $\mathcal{J}$. Thus, we have $\mathcal{S} \in \mathcal{F}_{\mathcal{J}}$. □

Corollary 6 *Given a job set $\mathcal{J}$, $E(\mathcal{S}^{\mathcal{J}}_{\text{opt}}) \leq E(\mathcal{S}^{\mathcal{J}^{\mathbf{f}}}_{\text{opt}})$ for any EDF-equivalent job set $\mathcal{J}^{\mathbf{f}}$. Furthermore, there exists an EDF-equivalent job set $\mathcal{J}^{\mathbf{f}}$ such that $\mathcal{S}^{\mathcal{J}}_{\text{opt}} \equiv \mathcal{S}^{\mathcal{J}^{\mathbf{f}}}_{\text{opt}}$.*

From Theorem 5, there is a one-to-one correspondence between feasible schedules of a fixed-priority job set $\mathcal{J}$ and feasible schedules of $\mathcal{J}$'s EDF-equivalent job sets. Since the energy-optimal schedule $\mathcal{S}^{\mathcal{J}^{\mathbf{f}}}_{\text{opt}}$ for an EDF-equivalent job set $\mathcal{J}^{\mathbf{f}}$ can be directly computed (in polynomial time) by Yao's algorithm [22], the problem of finding an energy-optimal (feasible) voltage schedule of $\mathcal{J}$ is reduced to the problem of finding an EDF-equivalent job set $\mathcal{J}^{\mathbf{f}}$ (or to selecting an EDF-ordered $|\mathcal{J}|$-tuple $\mathbf{f}$) that minimizes $E(\mathcal{S}^{\mathcal{J}^{\mathbf{f}}}_{\text{opt}})$.

Figure 1 shows an example of EDF-equivalent job sets and EDF-ordered $|\mathcal{J}|$-tuples. Figure 1.(a) shows the original job set $\mathcal{J} = \{J_1, J_2\}$. In this example, J_2 has a lower priority but earlier deadline than J_1, so $\mathcal{J}$ is not an EDF job set. (So Yao's algorithm cannot be directly applied to $\mathcal{J}$.) In Figures 1.(b) and 1.(c), two job sets are shown,



(a) (b) (c)

Figure 1. An example of EDF-equivalent job sets.

which are EDF-equivalent to $\mathcal{J}$. The job sets $\{J_1', J_2'\}$ and $\{J_1'', J_2''\}$ are obtained by choosing (r_{J_1}, d_{J_1}) and (d_{J_2}, d_{J_2}) as EDF-ordered $|\mathcal{J}|$-tuples, respectively. Both job sets follow the EDF priority assignment[1] and the optimal voltage schedule for each job set can be computed by Yao's algorithm. (As will be explained below, the energy-optimal voltage schedule of $\mathcal{J}$ is equal to $\mathcal{S}_{\text{opt}}^{\{J_1', J_2'\}}$ or $\mathcal{S}_{\text{opt}}^{\{J_1'', J_2''\}}$ depending on the workload of J_1 and J_2.)

Now we are to restrict the search space of EDF-ordered $|\mathcal{J}|$-tuples (equivalently, EDF-equivalent job sets). First, an EDF-ordered $|\mathcal{J}|$-tuple $\mathbf{f} = (f_1, f_2, \cdots, f_{|\mathcal{J}|})$ does not need to be considered if for another EDF-ordered $|\mathcal{J}|$-tuple $\mathbf{f}' = (f_1', f_2', \cdots, f_{|\mathcal{J}|}')$ $(\neq \mathbf{f})$, $f_i \leq f_i'$ for all $1 \leq i \leq |\mathcal{J}|$. This is because, for any voltage schedule $\mathcal{S}(t)$ which is feasible under $\mathbf{f}$, $\mathcal{S}(t)$ is also feasible under $\mathbf{f}'$. We define that an EDF-ordered $|\mathcal{J}|$-tuple $\mathbf{f}$ (or $\mathcal{J}^{\mathbf{f}}$) is *essential* if such $\mathbf{f}'$ does not exist. (The term 'essential' is equivalent to the term 'NAP' in [13].) Quan's optimal algorithm [13] finds an optimal voltage schedule by *exhaustively* enumerating all the essential (or NAP) job sets and then applying Yao's algorithm for each essential job set. Our fast algorithm avoids the exhaustiveness by carefully enumerating the essential job sets.

3.2 $|\mathcal{J}|$-Permutations

It is easy to check if a $|\mathcal{J}|$-tuple is EDF-ordered (or essential). On the contrary, it is not obvious how such $|\mathcal{J}|$-tuples can be enumerated. In this section, we describe how to construct EDF-ordered $|\mathcal{J}|$-tuples efficiently using a permutation-based analysis.

Given a $|\mathcal{J}|$-tuple $\mathbf{f} = (f_1, f_2, \cdots, f_{|\mathcal{J}|})$, let $\sigma_{\mathbf{f}} : \{1, 2, \cdots, |\mathcal{J}|\} \Rightarrow \{1, 2, \cdots, |\mathcal{J}|\}$ be a permutation that maps a new tuple index when the tuple elements are sorted in a non-decreasing order, i.e., $f_{\sigma_{\mathbf{f}}^{-1}(1)} \leq f_{\sigma_{\mathbf{f}}^{-1}(2)} \leq \cdots \leq f_{\sigma_{\mathbf{f}}^{-1}(|\mathcal{J}|)}$. Ties are broken by the priority, i.e., if $f_i = f_j$ where $i < j$, $\sigma_{\mathbf{f}}(i) < \sigma_{\mathbf{f}}(j)$. (From now on, we call such σ a $|\mathcal{J}|$-permutation.) For example, let $\mathbf{f} = (f_1, f_2, f_3, f_4) = (4, 10, 2, 10)$. Then, since $f_3 \leq f_1 \leq f_2 = f_4$, we have $\sigma(3) = 1$, $\sigma(1) = 2$, and (from the tie-breaking rule) $(\sigma(2), \sigma(4)) = (3, 4)$. (Equivalently, we have $(\sigma^{-1}(1), \sigma^{-1}(2), \sigma^{-1}(3), \sigma^{-1}(4)) = (3, 1, 2, 4)$.) Note that $\sigma^{-1}(i)$ denotes the index of the i-th smallest element in $\mathbf{f}$, i.e., $f_{\sigma^{-1}(i)}$ is the i-th smallest element in $\mathbf{f}$.

The following lemma states that there cannot exist more than one essential $|\mathcal{J}|$-tuples whose $|\mathcal{J}|$-permutations are the same, that is, each essential $|\mathcal{J}|$-tuple can be uniquely addressed by its corresponding $|\mathcal{J}|$-permutation (and, obviously, vice versa).

Lemma 7 *For any two essential $|\mathcal{J}|$-tuple $\mathbf{f} = (f_1, f_2, \cdots, f_{|\mathcal{J}|})$ and $\mathbf{f}' = (f_1', f_2', \cdots, f_{|\mathcal{J}|}')$ $(\mathbf{f} \neq \mathbf{f}')$, $\sigma_{\mathbf{f}} \neq \sigma_{\mathbf{f}'}$.*

Proof. Suppose $\sigma_{\mathbf{f}} \equiv \sigma_{\mathbf{f}'}$ and let i $(1 \leq i \leq |\mathcal{J}|)$ be the largest integer such that $f_{\sigma_{\mathbf{f}}^{-1}(i)} \neq f'_{\sigma_{\mathbf{f}'}^{-1}(i)}$, i.e.,

$$f_{\sigma_{\mathbf{f}}^{-1}(k)} = f'_{\sigma_{\mathbf{f}'}^{-1}(k)} \ (= f'_{\sigma_{\mathbf{f}}^{-1}(k)}) \quad \text{for all } i < k < |\mathcal{J}| \,. \tag{7}$$

Without loss of generality, we can assume $f_{\sigma_{\mathbf{f}}^{-1}(i)} < f'_{\sigma_{\mathbf{f}'}^{-1}(i)}$. Let us consider a new $|\mathcal{J}|$-tuple $\mathbf{f}'' = (f_1'', f_2'', \cdots, f_{|\mathcal{J}|}'')$ where

$$f_k'' = \begin{cases} f_k' & k = \sigma_{\mathbf{f}}^{-1}(i) \,, \\ f_k & \text{otherwise.} \end{cases}$$

From the definition of $\mathbf{f}''$, it can be easily seen that $\sigma_{\mathbf{f}''} \equiv \sigma_{\mathbf{f}} \equiv \sigma_{\mathbf{f}'}$. (We omit the subscripts in the rest of the proof.) We are now to prove that $\mathbf{f}''$ is EDF-ordered, i.e., for any $1 \leq j < k \leq |\mathcal{J}|$,

$$f_j'' \leq f_k'' \text{ or } f_k'' \leq r_{J_j} \,. \tag{8}$$

[1] In Figure 1.(c), J_1'' need not have an earlier deadline than J_2'' for the job set to be an EDF job set; $d_{J_1''} = d_{J_2''}$ is sufficient for the job set to be optimally scheduled by Yao's algorithm [22].

1: $f_{\sigma^{-1}(|\mathcal{J}|)} := d_{J_{\sigma^{-1}(|\mathcal{J}|)}}$
2: **for** ($i := |\mathcal{J}|-1$ to 1)
3: **let** $\mathcal{J}^{\mathrm{H}}$ be $\{J_{\sigma^{-1}(k)} \mid i < k \le |\mathcal{J}| \wedge \sigma^{-1}(k) < \sigma^{-1}(i)\}$
4: **if** ($r_{J_{\sigma^{-1}(i)}} \ge \min(\{r_J \mid J \in \mathcal{J}^{\mathrm{H}}\} \cup \{f_{\sigma^{-1}(i+1)}\})$) **return** FALSE
5: **else** $f_{\sigma^{-1}(i)} := \min(\{f_{\sigma^{-1}(i+1)}, d_{J_{\sigma^{-1}(i)}}\} \cup \{r_J \mid J \in \mathcal{J}^{\mathrm{H}}\})$
6: **end if**
7: **end for**

Figure 2. The algorithm to build a $|\mathcal{J}|$-tuple from a $|\mathcal{J}|$-permutation.

Since **f** is EDF-ordered, (8) holds for all $1 \le j < k \le |\mathcal{J}|$ except for $j = \sigma^{-1}(i)$ or $k = \sigma^{-1}(i)$. So, it remains to show that (8) holds for all $1 \le j < \sigma^{-1}(i) \le |\mathcal{J}|$ and $1 \le \sigma^{-1}(i) < k \le |\mathcal{J}|$.
Case (a): $1 \le j < \sigma^{-1}(i) \le |\mathcal{J}|$ (when J_j has a higher priority than $J_{\sigma^{-1}(i)}$.)
If $f''_j \le f''_{\sigma^{-1}(i)}$, (8) trivially holds. So, we only consider j such that $f''_j > f''_{\sigma^{-1}(i)}$, i.e., $f_j\ (= f_{\sigma^{-1}(\sigma(j))}) > f'_{\sigma^{-1}(i)}\ (> f_{\sigma^{-1}(i)})$. From the definition of σ, we have $\sigma(j) > i$. Thus, by substituting $\sigma(j)$ for k in Eq. (7), we have $f_j\ (= f''_j) = f'_j$. From the assumption, $\mathbf{f}'$ is EDF-ordered, but we have $f'_j = f_j > f'_{\sigma^{-1}(i)}$. So, it must be the case that $f'_{\sigma^{-1}(i)} \le r_{J_j}$. Therefore, we have

$$f''_{\sigma^{-1}(i)} = f'_{\sigma^{-1}(i)} \le r_{J_j}\ .$$

Case (b): $1 \le \sigma^{-1}(i) < k \le |\mathcal{J}|$ (when J_k has a lower priority than $J_{\sigma^{-1}(i)}$.)
First, we can exclude the case when $f_k = f_{\sigma^{-1}(i)}$. Otherwise, we have $\sigma(k) > \sigma(\sigma^{-1}(i)) = i$. (Recall the tie-breaking rule.) But, by the definition of σ, $f'_{\sigma^{-1}(\sigma(k))}\ (= f'_k) \ge f'_{\sigma^{-1}(i)}$ and we finally have

$$f'_k \ge f'_{\sigma^{-1}(i)} > f_{\sigma^{-1}(i)} = f_k\ ,$$

which contradicts Eq. (7).
Second, consider k such that $f_k < f_{\sigma^{-1}(i)}$. **f** is EDF-ordered, but we have $f_{\sigma^{-1}(i)} > f_k$. So, it must be the case that $f_k \le r_{J_{\sigma^{-1}(i)}}$. Therefore, we have

$$f''_k = f_k \le r_{J_{\sigma^{-1}(i)}}\ .$$

Finally, for k such that $f_k > f_{\sigma^{-1}(i)}$, we have

$$f''_{\sigma^{-1}(i)} = f'_{\sigma^{-1}(i)} \le f'_k = f_k = f''_k\ .$$

Thus, $\mathbf{f}''$ is EDF-ordered. However, since we have

$$f_{\sigma^{-1}(i)} < f'_{\sigma^{-1}(i)} = f''_{\sigma^{-1}(i)} \quad \text{and} \quad f_k = f''_k \quad \text{for all } 1 \le k \ne \sigma^{-1}(i) \le |\mathcal{J}|\ ,$$

f is not essential, a contradiction. Therefore, $\sigma_{\mathbf{f}} \ne \sigma_{\mathbf{f}'}$. □

The proof of Lemma 7 also implies how to build a unique essential job set for a σ.

Lemma 8 *Given a $|\mathcal{J}|$-permutation σ, the algorithm in Figure 2 finds a unique essential $|\mathcal{J}|$-tuple for σ if such a $|\mathcal{J}|$-tuple exists. Otherwise, it returns FALSE.*

Proof. First, suppose that the essential $|\mathcal{J}|$-tuple for σ exists and denote it by $\mathbf{f}' = (f'_1, f'_2, \cdots, f'_{|\mathcal{J}|})$. (Note that $f'_{\sigma^{-1}(1)} \le f'_{\sigma^{-1}(2)} \le \cdots \le f'_{\sigma^{-1}(|\mathcal{J}|)}$.) We are to prove that $f'_{\sigma^{-1}(i)} = f_{\sigma^{-1}(i)}$ and the algorithm does not abort in line 4 for all $i = |\mathcal{J}|, |\mathcal{J}|-1, \cdots, 1$ by induction on i. The base case holds trivially, i.e., $f'_{\sigma^{-1}(|\mathcal{J}|)} = d_{J_{\sigma^{-1}(|\mathcal{J}|)}} = f_{\sigma^{-1}(|\mathcal{J}|)}$. For the induction step, assume that the proposition holds for all $k = |\mathcal{J}|, |\mathcal{J}|-1, \cdots, i+1$. Let $\mathcal{J}^H = \{J_{\sigma^{-1}(k)} \mid i < k \le |\mathcal{J}| \wedge \sigma^{-1}(k) < \sigma^{-1}(i)\}$ (as in line 3 of the algorithm). Note that any job in $\mathcal{J}^H$ has the higher priority than $J_{\sigma^{-1}(i)}$ and that $f'_{\sigma^{-1}(i)} \le d_{J_{\sigma^{-1}(i)}}$ and $f'_{\sigma^{-1}(i)} \le f'_{\sigma^{-1}(i+1)}$.
Case (a): $\mathcal{J}^H = \emptyset$.
Suppose that $f'_{\sigma^{-1}(i)} < d_{J_{\sigma^{-1}(i)}}$ and $f'_{\sigma^{-1}(i)} < f'_{\sigma^{-1}(i+1)}$, that is,

$$f'_{\sigma^{-1}(1)} \le \cdots \le f'_{\sigma^{-1}(i)} < \min\{d_{J_{\sigma^{-1}(i)}},\ f'_{\sigma^{-1}(i+1)}\} \le f'_{\sigma^{-1}(i+1)} \le \cdots \le f'_{\sigma^{-1}(|\mathcal{J}|)}\ .$$

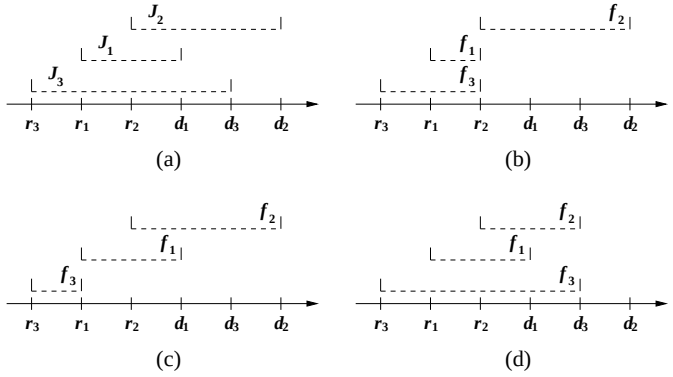


Figure 3. An example of $|\mathcal{J}|$-permutations. (a) A job set and its EDF-equivalent job sets for which $(\sigma^{-1}(3),\sigma^{-1}(2),\sigma^{-1}(1)) =$ (b) $(2,3,1)$, (c) $(2,1,3)$, and (d) $(3,2,1)$, respectively. ($(\sigma^{-1}(3),\sigma^{-1}(2),\sigma^{-1}(1)) = (1,2,3),(1,3,2)$ and $(3,1,2)$ are not valid $\mathcal{J}$-permutations.)

Let $\mathbf{f}'' = (f'_1,\cdots,f'_{\sigma^{-1}(i)-1},\min\{d_{J_{\sigma^{-1}(i)}},\ f'_{\sigma^{-1}(i+1)}\},f'_{\sigma^{-1}(i)+1},\cdots,f'_{|\mathcal{J}|})$. Then, $\mathbf{f}''$ is EDF-ordered, and $\mathbf{f}'$ is not essential, a contradiction. Therefore, we have

$$f'_{\sigma^{-1}(i)} \;=\; \min\{d_{J_{\sigma^{-1}(i)}},\ f'_{\sigma^{-1}(i+1)}\} \;=\; \min\{d_{J_{\sigma^{-1}(i)}},\ f_{\sigma^{-1}(i+1)}\} \;=\; f_{\sigma^{-1}(i)}\ .$$

Case (b): $\mathcal{J}^H \neq \emptyset$.
For all $J_{\sigma^{-1}(k)} \in \mathcal{J}^H$, we have $f'_{\sigma^{-1}(i)} < f'_{\sigma^{-1}(k)}$ from the definition of σ (Recall the tie-breaking rule.), and $f'_{\sigma^{-1}(i)} \leq r_{J_{\sigma^{-1}(k)}}$ since $\mathbf{f}'$ is EDF-ordered. Suppose that $f'_{\sigma^{-1}(i)} < \min\{r_J | J \in \mathcal{J}^{\mathrm{H}}\}$, $f'_{\sigma^{-1}(i)} < d_{J_{\sigma^{-1}(i)}}$ and $f'_{\sigma^{-1}(i)} < f'_{\sigma^{-1}(i+1)}$, that is,

$$f'_{\sigma^{-1}(1)} \leq \cdots \leq f'_{\sigma^{-1}(i)} < \min(\{d_{J_{\sigma^{-1}(i)}}, f'_{\sigma^{-1}(i+1)}\} \cup \{r_J | J \in \mathcal{J}^{\mathrm{H}}\}) \leq f'_{\sigma^{-1}(i+1)} \leq \cdots \leq f'_{\sigma^{-1}(|\mathcal{J}|)}\ .$$

Let $\mathbf{f}'' = (f'_1,\cdots,f'_{\sigma^{-1}(i)-1},\min(\{d_{J_{\sigma^{-1}(i)}}, f'_{\sigma^{-1}(i+1)}\} \cup \{r_J | J \in \mathcal{J}^{\mathrm{H}}\}),f'_{\sigma^{-1}(i)+1},\cdots,f'_{|\mathcal{J}|})$. Then, it can be easily shown that $\mathbf{f}''$ is EDF-ordered. Thus, $\mathbf{f}'$ is not essential, a contradiction. Therefore, we have

$$f'_{\sigma^{-1}(i)} \;=\; \min(\{d_{J_{\sigma^{-1}(i)}}, f'_{\sigma^{-1}(i+1)}\} \cup \{r_J | J \in \mathcal{J}^{\mathrm{H}}\}) \;=\; \min(\{d_{J_{\sigma^{-1}(i)}}, f_{\sigma^{-1}(i+1)}\} \cup \{r_J | J \in \mathcal{J}^{\mathrm{H}}\}) \;=\; f_{\sigma^{-1}(i)}\ .$$

Furthermore, we have for both cases

$$r_{J_{\sigma^{-1}(i)}} < f'_{\sigma^{-1}(i)} \leq \min(\{r_J | J \in \mathcal{J}^{\mathrm{H}}\} \cup \{f'_{\sigma^{-1}(i+1)}\}) = \min(\{r_J | J \in \mathcal{J}^{\mathrm{H}}\} \cup \{f_{\sigma^{-1}(i+1)}\})\ ,$$

and the algorithm does not abort in line 4 at iteration i, which completes the induction.

If the algorithm does not abort, the $|\mathcal{J}|$-tuple built by the algorithm is always a correct EDF-ordered $|\mathcal{J}|$-tuple, implying the existence of such $|\mathcal{J}|$-tuple for σ. Therefore, if such $|\mathcal{J}|$-tuple does not exist, the algorithm eventually returns FALSE. □

If a $|\mathcal{J}|$-permutation σ has the corresponding EDF-ordered $|\mathcal{J}|$-tuple $\mathbf{f}$, it is said to be *valid*. Furthermore, if $\mathbf{f}$ is essential, σ is said to be *essential*. From the above argument, we can establish one-to-one correspondences between EDF-ordered $|\mathcal{J}|$-tuples and valid $|\mathcal{J}|$-permutations, and between essential $|\mathcal{J}|$-tuples and essential $|\mathcal{J}|$-permutations. Figure 3.(a) shows a job set with three jobs and Figures 3.(b), 3.(c) and 3.(d) show its EDF-equivalent job sets with their $|\mathcal{J}|$-permutations. Among $3!(=6)$ possible $|\mathcal{J}|$-permutations, only three permutations are valid (and essential).

Based on the algorithm in Figure 2, we describe another way to enumerate $|\mathcal{J}|$-tuples. In the following, r_{J_i} and d_{J_i} are interpreted as symbolic values, not as real numbers. Then, $R_{\mathcal{J}} \cup D_{\mathcal{J}}$ has $2 \cdot |\mathcal{J}|$ distinct symbolic values. Furthermore, the algorithm in Figure 2 is assumed to assign symbolic values to elements of a $|\mathcal{J}|$-tuple with the following tie-breaking rule in line 5:

$$\text{(a) } r_{J_i} = r_{J_j}\ (i<j):\ \ r_{J_i} < r_{J_j} \quad \text{(b) } d_{J_i} = d_{J_j}\ (i<j):\ \ r_{J_i} < r_{J_j} \quad \text{(c) } r_{J_i} = d_{J_j}:\ \ r_{J_i} < d_{J_j}\ .$$

1: $\mathcal{J}' := \{\}, D := \{\}$
2: **foreach** ($d_{J_i} \in D_{\mathcal{J}}$ s.t. $\zeta(d_{J_i}) = 1$)
3: $f_i := d_{J_i}$, $\mathcal{J}' := \mathcal{J}' \cup \{J_i\}$, $D := D \cup \{d_{J_i}\}$
4: **end foreach** /* return FALSE here if $\mathcal{J}'$ does not follow the EDF priority. */
5: **foreach** ($r_{J_i} \in R_{\mathcal{J}}$ s.t. $\zeta(r_{J_i}) = 1$ in a decreasing order)
6: $f_i := \max\{d \in D \mid \mathcal{J}' \cup \{J\}$ follows the EDF priority where $p_J = p_{J_i}, r_J = r_{J_i}, d_J = d\}$
/* return FALSE here if such f_i does not exist. */
7: $\mathcal{J}' := \mathcal{J}' \cup \{J\}$, $D := D \cup \{r_{J_i}\}$
8: **end foreach**
9: **foreach** (J_i s.t. f_i is not determined (in any order))
10: $f_i := \max\{d \in D \mid \mathcal{J}' \cup \{J\}$ follows the EDF priority where $p_J = p_{J_i}, r_J = r_{J_i}, d_J = d\}$
/* return FALSE here if such f_i does not exist. */
11: $\mathcal{J}' := \mathcal{J}' \cup \{J\}$
12: **end foreach**

Figure 4. The algorithm to build a $|\mathcal{J}|$-tuple from a bit-vector.

Given a $|\mathcal{J}|$-tuple $\mathbf{f} = (f_1, f_2, \cdots, f_{|\mathcal{J}|})$, let $\zeta_{\mathbf{f}} : R_{\mathcal{J}} \cup D_{\mathcal{J}} \Rightarrow \{0,1\}$ be a bit-vector of length $2 \cdot |\mathcal{J}|$ such that

$$\zeta_{\mathbf{f}}(t) = \begin{cases} 1 & t = f_k \text{ for some } 1 \le k \le |\mathcal{J}| \ , \\ 0 & \text{otherwise.} \end{cases}$$

The algorithm in Figure 4 constructs a $|\mathcal{J}|$-tuple from an arbitrary bit-vector $\zeta : R_{\mathcal{J}} \cup D_{\mathcal{J}} \Rightarrow \{0,1\}$. The correctness of the algorithm can be proved in a similar manner as the algorithm in Figure 2.

3.3 An Alternative Formulation

The problem formulation given in Section 2 is based on the voltage schedule $\mathcal{S}(t)$. In this section, we describe an alternative formulation, based on the following intuitive property, which states that each job runs at the same constant speed if the voltage schedule is an optimal one.

Lemma 9 *For an energy-optimal voltage schedule $\mathcal{S}(t)$, $\mathcal{S}(t_1) = \mathcal{S}(t_2)$ for any t_1 and t_2 such that $job(\mathcal{J}, \mathcal{S}, t_1) = job(\mathcal{J}, \mathcal{S}, t_2)$.*

Proof. Given an optimal schedule $\mathcal{S}(t)$, suppose that $\mathcal{S}(t_1) \neq \mathcal{S}(t_2)$ for some t_1 and t_2 such that $job(\mathcal{J}, \mathcal{S}, t_1) = job(\mathcal{J}, \mathcal{S}, t_2)$. Given that $\mathcal{S}(t)$ is optimal, there exist t_1', t_2', S_1, S_2 and Δt such that $\mathcal{S}(t) = S_1$ for $t_1' \le t \le t_1' + \Delta t$, $\mathcal{S}(t) = S_2$ for $t_2' \le t \le t_2' + \Delta t$, and $S_1 \neq S_2$. Let $\mathcal{S}(t)'$ be defined by

$$\mathcal{S}(t)' = \begin{cases} \frac{S_1 + S_2}{2} & t_1' \le t \le t_1' + \Delta t \ , \ t_2' \le t \le t_2' + \Delta t, \\ \mathcal{S}(t) & \text{otherwise.} \end{cases}$$

Then, it is obvious that $\mathcal{S}(t)'$ is feasible and $E(\mathcal{S}') < E(\mathcal{S})$, a contradiction. □

From Lemma 9, it can be shown that the voltage scheduling problem is equivalent to determining the allowed execution time a_i allocated to each J_i. Given a feasible voltage schedule $\mathcal{S}$, the corresponding tuple of the allowed execution times $(a_1, a_2, \cdots, a_{|\mathcal{J}|})$, called a *time-allocation tuple,* can be uniquely determined. Conversely, given a time-allocation tuple $\mathbf{A} = (a_1, a_2, \cdots, a_{|\mathcal{J}|})$, the corresponding voltage schedule $\mathcal{S}_{\mathbf{A}}$ can be uniquely constructed by assigning the constant execution speed c_i / a_i to J_i. $\mathbf{A}$ is said to be *feasible* if the corresponding voltage schedule $\mathcal{S}_{\mathbf{A}}$ is feasible.

Let us now consider the exact condition for a time-allocation tuple $\mathbf{A} = (a_1, a_2, \cdots, a_{|\mathcal{J}|})$ to be feasible by rewriting Condition I in Section 2 in terms of $\mathbf{A}$.

Condition III (Feasibility Condition for Time-Allocation Tuples).
There exists a $|\mathcal{J}|$-tuple $(f_{J_1}, f_{J_2}, \cdots, f_{J_{|\mathcal{J}|}}) \in \mathcal{T}^{\mathcal{J}}$ such that

$$\forall 1 \le i \le |\mathcal{J}| \quad \forall r \in \{t \mid t \in R_{\mathcal{J}} \wedge t < f_{J_i}\} \quad \sum_{J_k / p_{J_k} \le p_{J_i} \wedge r_{J_k} \in [r, f_{J_i})} a_k \ \le \ f_{J_i} - r \ . \tag{9}$$

Lemma 10 *Condition III is a necessary and sufficient condition for* $\mathbf{A}$ *to be feasible.*

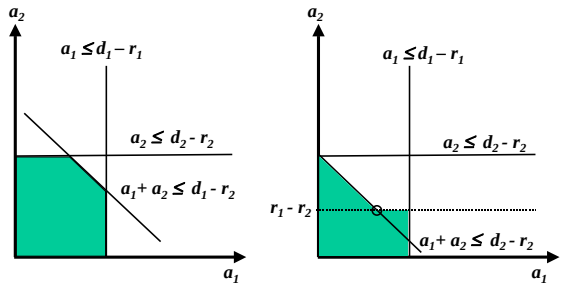


Figure 5. Solution spaces for (a) an EDF job set and (b) a fixed-priority job set.

Proof. Given a job set $\mathcal{J} = \{J_1, J_2, \cdots, J_{|\mathcal{J}|}\}$ and a time-allocation tuple $\mathbf{A} = (a_1, a_2, \cdots, a_{|\mathcal{J}|})$ for $\mathcal{J}$, consider a new job set $\mathcal{J}' = \{J'_1, J'_2, \cdots, J'_{|\mathcal{J}|}\}$ where $c_{J'_i} = a_i$, $r_{J'_i} = r_{J_i}$, $d_{J'_i} = d_{J_i}$, and $p_{J'_i} = p_{J_i}$ for all $1 \le i \le |\mathcal{J}|$, i.e., $\mathcal{J}'$ is identical to $\mathcal{J}$ except for the workload.

Let $\mathcal{S}'(t) = 1 \ \ (\forall t > 0)$ be the voltage schedule of $\mathcal{J}'$. Then, it is obvious that the response time of $\mathcal{J}_i$ under $\mathcal{S}_\mathbf{A}$ is the same as that of $\mathcal{J}'_i$ under $\mathcal{S}'$. Thus, $\mathbf{A}$ is feasible if and only if $\mathcal{S}'$ is a feasible voltage schedule for $\mathcal{J}'$. After replacing $\mathcal{S}$ and c_{J_i} in Condition I by $\mathcal{S}'$ and a_i, respectively, we have Condition III. □

By applying the same argument to Condition II, we have the following condition for EDF job sets.

Condition IV (EDF Feasibility Condition for Time-Allocation Tuples).

For any $r \in R_\mathcal{J}$ and $d \in D_\mathcal{J}$ (where $r < d$),

$$\sum_{J/[r_J, d_J] \subseteq [r,d]} a_i \ \le \ d - r \ .$$

Now, the voltage scheduling problem can be reformulated as follows:

Find a time-allocation tuple $\mathbf{A} = (a_1, a_2, \cdots, a_{|\mathcal{J}|})$ such that $E(\mathcal{S}_\mathbf{A})$ is minimized subject to Condition III (or Condition IV for an EDF job set).

The energy consumption of the voltage schedule $\mathcal{S}_\mathbf{A}$ can be directly computed:

$$E(\mathcal{S}_\mathbf{A}) \ = \ \sum_{i=1}^{|\mathcal{J}|} a_i \cdot P(c_i/a_i) \ . \tag{10}$$

The set of feasible time-allocation tuples represents the solution space for the voltage scheduling problem stated in terms of time-allocation tuples. For an EDF job set, the solution space is specified by a conjunction of linear inequalities which can be directly obtained from Condition IV. However, this is not the case for a fixed-priority job set; the existential quantifier in Condition III is not always removable. Consequently, the solution space for an EDF job set is a convex set while the solution space for an arbitrary fixed-priority job set may not be a convex set.

Before we present an intractability result for the voltage scheduling problem in the next section, we illustrate the inherent complexity of fixed-priority voltage scheduling based on the results explained in this section. Figures 5.(a) and 5.(b) show the solution spaces for an example EDF job set and an example fixed-priority job set, respectively. As a fixed-priority job set, we use the job set $\{J_1, J_2\}$ of Figure 1. As an EDF job set, we use the same job set $\{J_1, J_2\}$ in Figure 1 with the same timing parameters, but the priority assignment is changed such that it follows the EDF priority assignment, i.e., $p_{J_2} < p_{J_1}$. For the EDF job set, we have the following constraint:

$$a_1 \le d_{J_1} - r_{J_1} \ \wedge \ a_2 \le d_{J_2} - r_{J_2} \ \wedge \ a_1 + a_2 \le d_{J_1} - r_{J_2}$$

Similarly, we have the following constraint for the fixed-priority job set:

$$a_1 \le d_{J_1} - r_{J_1} \ \wedge \ a_2 \le r_{J_1} - r_{J_2} \quad \text{(Figure 1.(b))} \quad \vee \quad a_1 \le d_{J_2} - r_{J_1} \ \wedge \ a_1 + a_2 \le d_{J_2} - r_{J_2} \quad \text{(Figure 1.(c))}$$

In Figures 5.(a) and 5.(b), the solution spaces for the EDF job set and the fixed-priority job set are depicted as a convex region and a concave region, respectively. (Each point in the shaded regions represents a feasible schedule.)

In general, the solution space of any EDF job set with N jobs are represented by a convex set in $\mathbf{R}^N$, whereas the solution space of a fixed-priority job set is represented by a concave set. Note that for EDF job sets, the objective function, the total energy consumption, can be efficiently minimized by an optimization technique for a convex set (as in Yao's algorithm). However, optimization problems defined on a concave set are generally intractable.

4 Intractability Result

In this section, we present some observations related to the complexity issue of the optimal fixed-priority scheduling problem. We first show that the decision version of the problem is NP-hard.

Theorem 11 *Given a job set $\mathcal{J}$ and a positive number K, the problem of deciding if there is a feasible voltage schedule $\mathcal{S}(t)$ for $\mathcal{J}$ such that $E(\mathcal{S}) \leq K$ is NP-hard.*

Proof. Without loss of generality, we assume that the energy consumption (per CPU cycle) is quadratically dependent on the processor speed. That is, the instantaneous power consumption (per time) is cubically dependent on the processor speed, i.e., $P(t) = \mathcal{S}(t)^3$. (The reduction can be easily modified for other power functions.) We prove the theorem by reduction from the subset-sum problem, which is NP-complete [3]:

SUBSET-SUM

INSTANCE: A finite set $\mathbf{U}$, a size $s : \mathbf{U} \Rightarrow \mathbf{Z}^+$, and a positive integer B.
Question: Is there a subset $\mathbf{U}' \subseteq \mathbf{U}$ such that $\sum_{u \in \mathbf{U}'} s(u) = B$?

Given an instance $\langle \mathbf{U} (= \{u_1, \cdots, u_{|\mathbf{U}|}\}), s, B \rangle$ of the subset-sum problem, we construct a job set $\mathcal{J}$ and a positive number K such that there is a voltage schedule $\mathcal{S}(t)$ of $\mathcal{J}$ with $E(\mathcal{S}(t)) \leq K$ if and only if $\exists \mathbf{U}' \subseteq \mathbf{U}$, $\sum_{u \in \mathbf{U}'} s(u) = B$. The corresponding job set $\mathcal{J}$ consists of $2 \cdot |\mathbf{U}| + 1$ jobs as follows:

$$
\begin{aligned}
\mathcal{J} \;=\; & \{J_1, J_2, \cdots, J_{2\cdot|\mathbf{U}|+1}\} \qquad \text{where} \\
& p_{J_i} \;=\; i \quad \text{for all } \; 1 \leq i \leq 2 \cdot |\mathbf{U}| + 1 \,, \\
& r_{J_{2\cdot i+1}} \;=\; s(u_{i+1}) \;+\; \sum_{j=1}^{i} 3 \cdot s(u_j) \;\;,\;\; r_{J_{2\cdot i+2}} \;=\; \sum_{j=1}^{i} 3 \cdot s(u_j) \;\;, \\
& d_{J_{2\cdot i+1}} \;=\; \sum_{j=1}^{i+1} 3 \cdot s(u_j) \;\;,\;\; d_{J_{2\cdot i+2}} \;=\; 2 \cdot s(u_{i+1}) \;+\; \sum_{j=1}^{i} 3 \cdot s(u_j) \;\;, \\
& c_{J_{2\cdot i+1}} \;=\; 8 \cdot \gamma \cdot s(u_{i+1}) \;\;,\;\; c_{J_{2\cdot i+2}} \;=\; 8 \cdot s(u_{i+1}) \quad \text{for all } \; 0 \leq i \leq |\mathbf{U}| - 1 \,, \text{ and} \\
& r_{J_{2\cdot|\mathbf{U}|+1}} \;=\; 0 \;\;,\;\; d_{J_{2\cdot|\mathbf{U}|+1}} \;=\; B \;+\; \sum_{j=1}^{i} 3 \cdot s(u_j) \;\;,\;\; c_{J_{2\cdot|\mathbf{U}|+1}} \;=\; \sqrt[3]{4} \cdot B \;\;.
\end{aligned}
$$

where γ is the unique positive solution of the following quadratic equation:

$$\gamma^2 + \gamma = 1 + \frac{4}{3 \cdot 8^3} \qquad (\Longrightarrow \; \frac{1}{2} \;<\; \gamma \;<1).$$

Furthermore, K is set to be

$$K \;=\; (8^3 + \frac{\gamma^3}{4} \cdot 8^3) \cdot \sum_{i=1}^{|\mathbf{U}|} s(u_i) \;\; + 2 \cdot B \,.$$

From the construction of $\mathcal{J}$, we have

$$
\begin{aligned}
& r_{J_{2\cdot i+2}} < r_{J_{2\cdot i+1}} \,(= r_{J_{2\cdot i+2}} + s(u_{i+1})) < d_{J_{2\cdot i+2}} \,(= r_{J_{2\cdot i+1}} + s(u_{i+1})) < d_{J_{2\cdot i+1}} \,(= d_{J_{2\cdot i+2}} + s(u_{i+1})) \;\;, \\
& [r_{J_{2\cdot i+2}}, d_{J_{2\cdot i+1}}] \;\subset\; [r_{J_{2\cdot|\mathbf{U}|+1}}, d_{J_{2\cdot|\mathbf{U}|+1}}] \quad \text{for all } \; 0 \leq i \leq |\mathbf{U}| - 1 \quad \text{and} \\
& [r_{J_{2\cdot i+2}}, d_{J_{2\cdot i+1}}] \cap [r_{J_{2\cdot i'+2}}, d_{J_{2\cdot i'+1}}] \;=\; \emptyset \quad \text{for all } 0 \leq i \neq i' \leq |\mathbf{U}| - 1 \,.
\end{aligned}
$$

Let $\kappa : \{0,1\}^{|\mathbf{U}|} \;\Rightarrow\; \mathcal{T}^{\mathcal{J}}$ be a function defined by

$$
\begin{aligned}
\kappa((b_1, b_2, \cdots, b_{|\mathbf{U}|}) \;) \;=\; & (f_1, f_2, \cdots, f_{|\mathcal{J}|}) \quad \text{where} \\
& f_{2\cdot i+1} \;=\; d_{J_{2\cdot i+1}} \;\;,\;\; f_{2\cdot i+2} \;=\; r_{J_{2\cdot i+1}} \quad \text{if } b_{i+1} = 0 \;\;, \\
& f_{2\cdot i+1} \;=\; f_{2\cdot i+2} \;=\; d_{J_{2\cdot i+2}} \qquad\quad \text{if } b_{i+1} = 1 \quad \text{for all } 0 \leq i \leq |\mathbf{U}| - 1 \,, \text{ and} \\
& f_{2\cdot|\mathbf{U}|+1} \;=\; d_{J_{2\cdot|\mathbf{U}|+1}} \,.
\end{aligned}
$$

Then, the set of essential job sets of $\mathcal{J}$ is given by:

$$\{\mathcal{J}^{\mathbf{f}} \mid \mathbf{f} = \kappa(\mathbf{b}) \ , \ \mathbf{b} \in \{0,1\}^{|\mathbf{U}|}\}$$

To compute the energy consumption of an essential job set by Yao's algorithm [22], we first compare the intensity of each interval. Let

$$\begin{aligned} I_1 &= \frac{c_{J_{2\cdot i+2}}}{r_{J_{2\cdot i+1}} - r_{J_{2\cdot i+2}}} \ , \ I_2 = \frac{c_{J_{2\cdot i+1}}}{d_{J_{2\cdot i+1}} - r_{J_{2\cdot i+1}}} \ , \\ I_3 &= \frac{c_{J_{2\cdot i+1}}}{d_{J_{2\cdot i+2}} - r_{J_{2\cdot i+1}}} \ , \ I_4 = \frac{c_{J_{2\cdot i+1}} + c_{J_{2\cdot i+2}}}{d_{J_{2\cdot i+2}} - r_{J_{2\cdot i+2}}} \quad \text{and } I_5(\delta) = \frac{c_{J_{2\cdot|\mathbf{U}|+1}}}{B+\delta} \ . \end{aligned}$$

Then, we have

$$\begin{aligned} &I_1 = \frac{8\cdot s(u_{i+1})}{s(u_{i+1})} = 8 \ > \ I_2 = \frac{8\cdot\gamma\cdot s(u_{i+1})}{2\cdot s(u_{i+1})} = 4\cdot\gamma \ > \ 2 \ > \ \sqrt[3]{4} \ > \ I_5 = \frac{\sqrt[3]{4}\cdot B}{B+\delta} \quad \text{and} \\ &I_4 = \frac{8\cdot(1+\gamma)\cdot s(u_{i+1})}{2\cdot s(u_{i+1})} = 4+4\cdot\gamma \ > \ I_3 = \frac{8\cdot\gamma\cdot s(u_{i+1})}{s(u_{i+1})} = 8\cdot\gamma \ > \ I_5 \ . \end{aligned}$$

So, the energy consumption of $\mathcal{S}_{\text{opt}}^{\mathcal{J}^{\mathbf{f}}}$ for $\mathbf{f} = \kappa((b_1, b_2, \cdots, b_{|\mathbf{U}|}))$ can be computed as follows:

$$E(\mathcal{S}_{\text{opt}}^{\mathcal{J}^{\mathbf{f}}}) \ = \ \sum_{i=1}^{|\mathbf{U}|} E_i \ + \ E_{\text{L}} \qquad \text{where}$$

$$E_i = \begin{cases} (8^3 + \frac{\gamma^3}{4}\cdot 8^3)\cdot s(u_i) \ \ (= \frac{(8\cdot s(u_i))^3}{s(u_i)^2} + \frac{(8\cdot\gamma\cdot s(u_i))^3}{(2\cdot s(u_i))^2}) & b_i = 0 \ , \\ \frac{(1+\gamma)^3}{4}\cdot 8^3\cdot s(u_i) \ \ (= \frac{(8\cdot(1+\gamma)\cdot s(u_i))^3}{(2\cdot s(u_i))^2}) & b_i = 1 \qquad \text{and} \end{cases}$$

$$E_{\text{L}} \ = \ \frac{4\cdot B^3}{(B \ + \ \sum_{i=1}^{|\mathbf{U}|} b_i\cdot s(u_i))^2} \quad \left(= \frac{c^3_{J_{2\cdot|\mathbf{U}|+1}}}{(B \ + \ \sum_{i=1}^{|\mathbf{U}|} b_i\cdot(d_{J_{2\cdot i-1}} - d_{J_{2\cdot i}}))^2} \right) \ .$$

Since we have

$$\begin{aligned} \frac{(1+\gamma)^3}{4}\cdot 8^3\cdot s(u_i) &= \frac{1+3\cdot\gamma+3\cdot\gamma^2+\gamma^3}{4}\cdot 8^3\cdot s(u_i) \\ &= \frac{1+3\cdot(1+4/(3\cdot 8^3))+\gamma^3}{4}\cdot 8^3\cdot s(u_i) \ = \ (8^3+\frac{\gamma^3}{4}\cdot 8^3)\cdot s(u_i) \ + \ s(u_i) \ , \end{aligned}$$

we can rewrite $E(\mathcal{S}_{\text{opt}}^{\mathcal{J}^{\mathbf{f}}})$ as follows:

$$E(\mathcal{S}_{\text{opt}}^{\mathcal{J}^{\mathbf{f}}}) \ = \ (8^3+\frac{\gamma^3}{4}\cdot 8^3)\cdot\sum_{i=1}^{|\mathbf{U}|} s(u_i) \ + \ x \ + \ \frac{4\cdot B^3}{(B \ + \ x)^2} \quad \text{where} \quad x \ = \ \sum_{i=1}^{|\mathbf{U}|} b_i\cdot s(u_i) \ .$$

It can be easily shown that $E(\mathcal{S}_{\text{opt}}^{\mathcal{J}^{\mathbf{f}}})$ has the minimum $(8^3+\frac{\gamma^3}{4}\cdot 8^3)\cdot\sum_{i=1}^{|\mathbf{U}|} s(u_i) \ + \ 2\cdot B \ \ (= K)$ at $x = B$. That is, $E(\mathcal{S}_{\text{opt}}^{\mathcal{J}^{\mathbf{f}}}) \le K$ if and only if

$$\begin{aligned} &\exists\,(b_1, b_2, \cdots, b_{|\mathbf{U}|}) \in \{0,1\}^{|\mathbf{U}|} \ , \quad \textstyle\sum_{i=1}^{|\mathbf{U}|} b_i\cdot s(u_i) \ = \ B \ , \text{ which is equivalent to} \\ &\exists\,\mathbf{U}' \in \mathbf{U} \ , \ \sum_{u\in\mathbf{U}'} s(u) \ = \ B \ . \end{aligned}$$

It is obvious that the transformation can be done in polynomial time. Therefore, the problem is NP-hard. □

From the NP-hardness proof, the problem seems unlikely to have polynomial time algorithms that compute optimal solutions. The NP-hardness of the problem strongly depends on the fact that extremely large input numbers are allowed, as with some other NP-hard problems (e.g., the subset-sum problem and the knapsack problem [3]). The NP-hardness in the ordinary (but not strong) sense does not rule out possibility of existence of a pseudo-polynomial time algorithm or an FPTAS. Since our problem is an optimization problem that handles real numbers, we focus our attention on the FPTAS in the next section.

5 A Fast Approximation Scheme

In this section, we present a fully polynomial time approximation scheme (FPTAS) for the problem. We first consider a dynamic programming formulation that always finds the optimal solution, but may run in exponential time. Then, the dynamic programming formulation is transformed into an FPTAS by using a standard technique, the *rounding-the-input-data* technique [21]. The technique brings the running time of the dynamic program down to polynomial by rounding the input data so that sufficiently close input data are treated by a representative data [14]. The relative error of an approximation scheme depends on how we define the closeness; the smaller the threshold value for the closeness is, the smaller the relative error is. For a smaller error bound, however, the computation time becomes longer.

5.1 Algorithm for Optimal Solutions

We first present an exponential-time optimal algorithm based on the properties of optimal voltage schedules described in Section 3. The exponential-time algorithm essentially enumerates all the essential job sets. However, unlike Quan's exhaustive algorithm [13], it enumerates the essential job sets intelligently without actually enumerating all of them. Furthermore, it is based on dynamic programming formulation so that it can be easily transformed into an FPTAS by the standard technique.

In formulating the problem by dynamic programming, we first identify appropriate 'overlapping' (or reusable) subproblems to which dynamic programming can be applied iteratively. We note that the 'optimal substructure' of our problem is naturally reflected by *blocking tuples*, which are just sequences of time points in $\mathcal{T}_\mathcal{J}$ in strictly increasing order. (We formally define the blocking tuples later in this section.) That is, the optimal solution of the original problem can be built by just merging the optimal schedules of the sub-intervals defined by a blocking tuple. Figure 6 shows an example job set and its corresponding EDF-equivalent job set whose time interval is partitioned by a blocking tuple $(r_N, r_{N-3}, d_{N-1}, \cdots, r_2, d_2)$, which is depicted by a set of the dashed thick lines in Figure 6.(b). Note that jobs in each sub-interval follow the EDF-priority assignment.

The original problem is partitioned into subproblems by partitioning the overall time interval into sub-intervals such that jobs in each sub-interval follow the EDF priority assignment. If a job is released within a sub-interval with its deadline outside the sub-interval, the deadline can be modified to the end of the sub-interval. Each partitioned interval can be optimally scheduled in polynomial time by Yao's algorithm [22]. The challenge is how to find the set of sub-intervals whose optimal sub-schedules build an energy-optimal voltage schedule.

5.1.1 Basic Idea: The First Example

We now explain the basic idea of the optimal algorithm by describing the optimal algorithm on a simple but illustrative job set $\mathcal{J} = \{J_1, J_2, \cdots, J_N\}$ in Figure 6.(a) where $r_{i+1} < r_i < d_{i+1} < d_i$ for $1 \le i < N$. (Note that if the priorities of jobs are reversed, the job set follows the EDF priority.) For this job set, an essential job set $\mathcal{J}^{\mathrm{e}}$ (such as one in Figure 6.(b)) is partitioned into $\mathcal{J}_1^{\mathrm{e}}, \mathcal{J}_2^{\mathrm{e}}, \cdots, \mathcal{J}_k^{\mathrm{e}}$ such that each $\mathcal{J}_i^{\mathrm{e}}$ $(1 \le i \le k)$ follows the EDF priority assignment and the union I_i of execution intervals of jobs in $\mathcal{J}_i^{\mathrm{e}}$ (i.e., $I_i = \cup_{J \in \mathcal{J}_i^{\mathrm{e}}}[r_J, d_J]$) does not overlap with I_j $(= \cup_{J' \in \mathcal{J}_j^{\mathrm{e}}}[r_{J'}, d_{J'}])$ for all $1 \le i \ne j \le k$. To be more concrete,

$$\text{for all } 1 \le i < j < k\,,\ \forall J \in \mathcal{J}_i^{\mathrm{e}}, J' \in \mathcal{J}_j^{\mathrm{e}}\,,\ d_J \le r_{J'}\,.$$

Therefore, the optimal voltage schedule $\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}^{\mathrm{e}}}$ of $\mathcal{J}^{\mathrm{e}}$ is equal to the concatenation of the optimal voltage schedules of $\mathcal{J}_i^{\mathrm{e}}$'s, i.e.,

$$\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}^{\mathrm{e}}}(t) \equiv \oplus_{i=1}^{k} \mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_i^{\mathrm{e}}}(t)\,.$$

Note that $\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_i^{\mathrm{e}}}$ can be directly computed by Yao's algorithm [22] since $\mathcal{J}_i^{\mathrm{e}}$ follows the EDF priority assignment. Therefore, the energy-optimal fixed-priority voltage scheduling problem is further reduced to the problem of finding a partition that gives the energy-optimal voltage schedule for the whole time interval.

In defining a partition, we use a blocking tuple. For example, assume that f_N is selected as r_{N-3} as in Figure 6.(b). Then, both f_{N-1} and f_{N-2} should be selected as r_{N-3}, so that the job set becomes EDF-equivalent and, furthermore, essential. As shown in Figure 6.(b), these three jobs are separated from the other jobs by a thick vertical line at time r_{N-3}. These jobs constitutes the first partitioned job set $\mathcal{J}_1^{\mathrm{e}}$. The remaining job sets $\mathcal{J}_2^{\mathrm{e}}, \cdots, \mathcal{J}_k^{\mathrm{e}}$

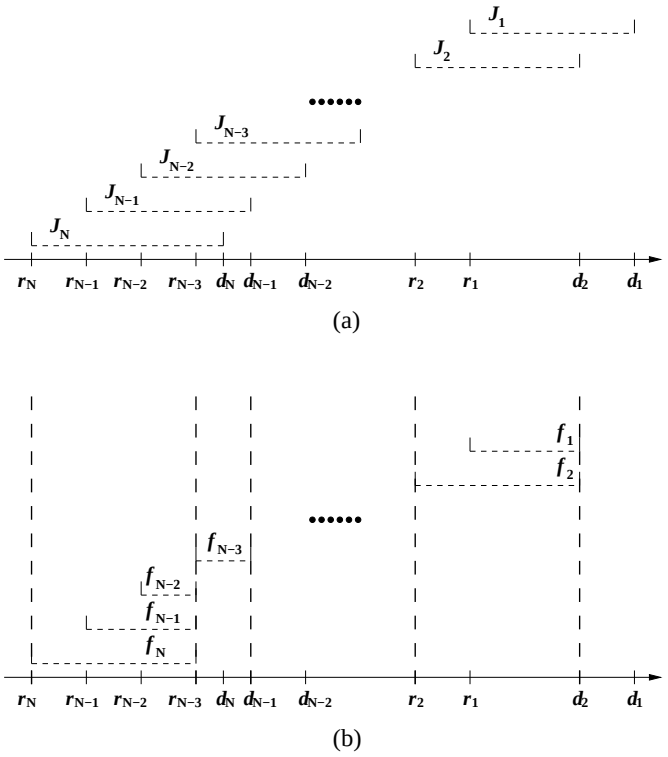


Figure 6. An example illustrating the optimal algorithm. (a) An original job set and (b) an essential job set defined by a $|\mathcal{J}|$-tuple $\mathbf{f} = (f_1, f_2, \cdots, f_{N-3}, f_{N-2}, f_{N-1}, f_N) = (d_2, d_2, \cdots, d_{N-1}, r_{N-3}, r_{N-3}, r_{N-3})$. Jobs in each sub-interval between the thick dashed lines follows the EDF priority assignment and can be optimally scheduled by Yao's algorithm.

can be constructed by applying the same argument. In this way, any essential job set can be partitioned and represented by a blocking tuple.

Let $\mathbf{b} = (b_1, b_2, \cdots, b_l)$ $(b_1 < b_2 < \cdots < b_l\ ,\ b_j \in T_{\mathcal{J}})$ be a blocking tuple where

$$\forall 1 \le j < l\ ,\ \exists J_i \text{ s.t. } b_j = r_i \wedge b_{j+1} \le d_i$$

Then, the corresponding EDF-ordered $|\mathcal{J}|$-tuple $\mathbf{f} = (f_1, f_2, \cdots, f_N)$ is given by

$$f_k = b_j \text{ s.t. } r_k \in [b_{j-1}, b_j) \text{ for all } 1 \le k \le N\ .$$

We call such $[b_{j-1}, b_j]$ an *atomic interval*. For example, the intervals $[r_N, r_{N-3}]$ and $[r_N, d_N]$ in Figure 6.(a) are atomic, but the interval $[r_N, d_{N-1}]$ is not atomic. (Later, we formally define the term atomic interval in arbitrary job sets other than this example.) Let t_h be the h-th earliest time point in $T_{\mathcal{J}}$ and let $\mathcal{S}_{h,g}$ represent the energy-optimal voltage schedule defined within $[t_h, t_g]$ for the job set $\mathcal{J}_{h,g}$ defined by

$$\mathcal{J}_{h,g} = \{J_i' |\ r_{J_i} \in [t_h, t_g)\} \text{ where } r_{J_i'} = r_{J_i}, c_{J_i'} = c_{J_i}, p_{J_i'} = p_{J_i} \text{ and } d_{J_i'} = \min\{d_{J_i}, t_g\}\ .$$

Then, we have

$$E(\mathcal{S}^{\mathcal{J}}_{\text{opt}}) = E(\mathcal{S}_{1,|T_{\mathcal{J}}|}) = \min\{\sum_{j=1}^{k-1} E(\mathcal{S}_{h_j,h_{j+1}}) \mid 1 = h_1 < h_2 < \cdots < h_k = |\mathcal{J}| \text{ and } [t_{h_j}, t_{h_{j+1}}] \text{ is atomic for all } j = 1, \cdots, k-1\}\ .$$

Given an atomic interval $[t_{h_j}, t_{h_{j+1}}]$, $\mathcal{S}_{h_j,h_{j+1}}$ can be directly computed by Yao's algorithm. In this way, the optimal voltage scheduling problem is reduced to a variant of the subset-sum problem. That is, for such job sets as in Figure 6, our problem can be formulated as follows:

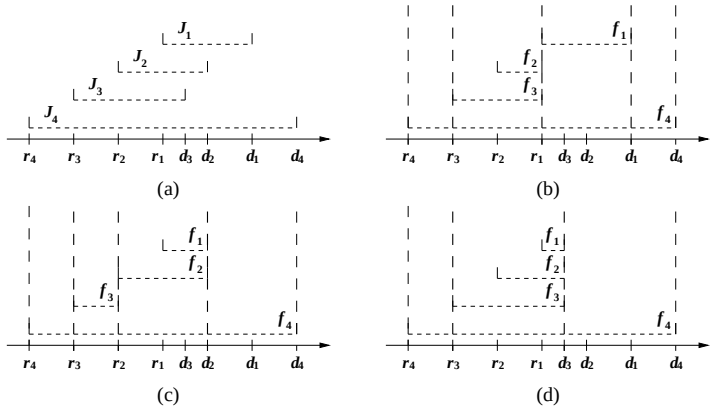


Figure 7. An example of background workload.

Select a tuple $(h_1, h_2, \cdots, h_k)$ $(1 = h_1 < \cdots < h_k = |\mathcal{J}|)$ of integers such that the sum

$$q_{h_1,h_2} + q_{h_2,h_3} + \cdots + q_{h_{k-1},h_k}$$

is minimized subject to $[t_{h_i}, t_{h_{i+1}}]$ is atomic for all $1 \leq i < k$ where $q_{h_j,h_{j+1}}$ denotes $E(\mathcal{S}_{h_j,h_{j+1}})$ (which can be directly computed by Yao's algorithm).

5.1.2 Basic Idea: The Second Example

The example job set in Figure 6 is illustrative in showing how our problem can be formulated by dynamic programming. However, the easily partitionable structure comes from the fact the job set follows the 'reverse' EDF priority. For example, in Figure 6, since f_N is set to be r_{N-3}, which is within the execution intervals of J_{N-1} and J_{N-2}, f_{N-1} and f_{N-2} cannot be larger than f_N (or r_{N-3}) so that the modified job set should be EDF-equivalent. Furthermore, f_{N-1} and f_{N-2} are set to be the maximum possible value, f_N, for the modified job set to be essential.

If the priority pattern is not the same as the example job set in Figure 6, the partitioning becomes difficult. For example, the essential job sets in Figures 3.(c) and 3.(d) cannot be obtained by the partitioning procedure just explained. In Figure 7.(a), J_4 has the lowest priority and the latest deadline, which makes f_4 to be d_4 for all essential job sets (Figures 7.(a), 7.(b) and 7.(c)). Therefore, any atomic interval (e.g., $[r_3, r_1]$, $[r_1, d_1]$ or $[r_3, d_3]$) contains partial workload of J_4, which we call a *background* workload. In the following, we first explain how to extend the dynamic programming formulation to handle the *background* workload. Then, we describe how to explore essential job sets of a given arbitrary job set (as in Figure 3) by dynamic programming.

From Lemma 9, the job J_4 in Figure 7 runs at the same speed if the voltage schedule is an optimal one. For the time being, let us assume that the constant speed is among $S_C = \{s_1, s_2, s_3\}$. (For now, S_C is set to be the set of all the possible constant speeds in the optimal voltage schedule. In section 5.2, we explain how the set S_C is selected such that the size of S_C is bounded by a polynomial function.) For each $s_i \in S_C$, we first compute the amount of background workload of J_4 for each atomic interval, and then find the minimum-energy essential job set (among those in Figures 7.(b), 7.(c) and 7.(d)) by using the similar procedure to the previous case in Figure 6. However, unlike the previous case, we discard any job set for which the sum of background workloads executed in overall time interval is less than the total workload of J_4.

Figure 8.(a) shows the atomic intervals $[r_3, r_1]$ and $[r_1, d_1]$, which are obtained from the essential job set in Figure 7.(b). Figures 8.(b), 8.(c) and 8.(d) show the optimal voltage schedules for the atomic intervals where J_4 runs at the speed s_1, s_2 and s_3, respectively. The workloads of jobs J_1, J_2 and J_3 are denoted by c_1, c_2 and c_3, respectively, and the background workloads are denoted by w. The amount of the background workload (and the resultant optimal voltage schedule) for each atomic interval and speed can be easily computed by a slightly modified version of Yao's algorithm [22]. That is, when the critical interval is selected, if the speed to be assigned (by the intensity of the critical interval) is less than or equal to the speed of the background workload, we assign the speed of the background workload to all the unscheduled time intervals (including the critical interval). Then, the amount of background workload can be directly computed as in Figure 8.(b), 8.(c) and 8.(d).

Once the background workload and the optimal voltage schedule is computed for each atomic interval, we apply the same procedure as in the job set in Figure 6 to find the minimum-energy essential job set and the energy-optimal

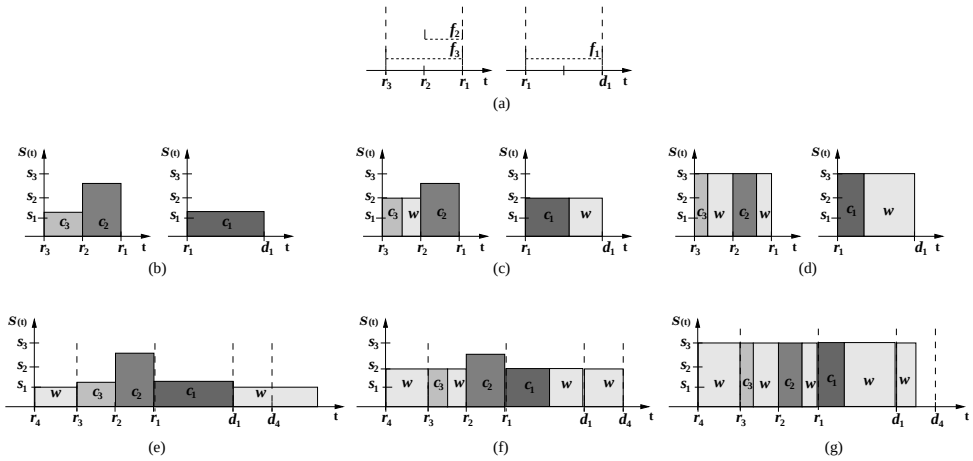


Figure 8. An example illustrating the algorithm on a job set with background workload. (a) Atomic intervals (obtained from the job set in Figure 7.(b)). The optimal schedules for two atomic intervals where the speeds of background workload of J_4 are (b) s_1, (c) s_2 and (d) s_3, respectively. The voltage schedules for overall time intervals where the speeds of J_4 are (e) s_1, (f) s_2 and (g) s_3, respectively.

1: $f_{\sigma^{-1}(|\mathcal{J}|)} := d_{J_{\sigma^{-1}(|\mathcal{J}|)}}$
2: $\mathbf{b}_\sigma := (d_{J_{\sigma^{-1}(|\mathcal{J}|)}})$
3: **for** ($i := |\mathcal{J}|-1$ to 1)
4: **let** $\mathcal{J}^{\mathrm{H}}$ be $\{J_{\sigma^{-1}(k)} | \; i < k \le |\mathcal{J}| \wedge \sigma^{-1}(k) < \sigma^{-1}(i)\}$
5: **if** ($r_{J_{\sigma^{-1}(i)}} \ge \min(\{r_J | J \in \mathcal{J}^{\mathrm{H}}\} \cup \{f_{\sigma^{-1}(i+1)}\})$) **return** FALSE
6: **else** $f_{\sigma^{-1}(i)} := \min(\{f_{\sigma^{-1}(i+1)}, d_{J_{\sigma^{-1}(i)}}\} \cup \{r_J | J \in \mathcal{J}^{\mathrm{H}}\})$
7: **end if**
8: **if** ($f_{\sigma^{-1}(i)} \le \min\{r_{J_{\sigma^{-1}(k)}} | \; i < k \le |\mathcal{J}|\}$) append $f_{\sigma^{-1}(i)}$ onto the head of $\mathbf{b}_\sigma$
9: **end if**
10: **end for**
11: append min $R_{\mathcal{J}}$ onto the head of $\mathbf{b}_\sigma$

Figure 9. The algorithm to build a strongly-blocking tuple from a $|\mathcal{J}|$-permutation.

voltage schedule. In exploring the solution space, we should discard any infeasible schedules. Figure 8.(e) shows an infeasible schedule where J_4 runs at s_1 and cannot complete its execution until its deadline. The voltage schedule in Figure 8.(g) is feasible, but not an optimal one. Thus, only the schedule in Figure 8.(f) is not removed in the pruning procedure and is compared with another schedules obtained from the essential job sets in Figures 8.(c) and 8.(d).

5.1.3 Putting It Altogether

We now describe the optimal algorithm for arbitrary job sets based on the observations from the example job sets. First, we formally define the terms *strongly-atomic interval* and *strongly-blocking tuple*. Given a valid $|\mathcal{J}|$-permutation σ, the algorithm in Figure 9 builds the corresponding strongly-blocking tuple $\mathbf{b}_\sigma = (b_1, b_2, \cdots, b_k)$ where $b_1 < b_2 < \cdots < b_k$ and $b_i \in T_{\mathcal{J}}$ for all $1 \le i \le k$. The algorithm is identical to the algorithm in Figure 2 except for lines 2, 8, 9 and 11. In line 8, $f_{\sigma^{-1}(i)}$ is selected as an element of a strongly-blocking tuple if it partitions the execution interval.

Definition 12 *Given a valid $|\mathcal{J}|$-permutation σ, the tuple $\mathbf{b}_\sigma$ built by the algorithm in Figure 9 is called a* strongly-blocking tuple. *An interval $[t,t']$ is* strongly-atomic *if there is a strongly-blocking tuple $\mathbf{b} = (b_1, b_2, \cdots, b_k)$ such that $[t,t'] = [b_i, b_{i+1}]$ for some $1 \le i < k$. Furthermore, the job set $\mathcal{J}_{[t,t']}$ defined by*

$$\mathcal{J}_{[t,t']} = \{J' | \; J \in \mathcal{J} \;,\; r_J \in [t,t')\} \;\; \textit{where} \;\; r_{J'} = r_J, c_{J'} = c_J, p_{J'} = p_J \;\; \textit{and} \;\; d_{J'} = \min\{d_J, t'\} \;.$$

is said to be induced *by an interval $[t,t']$.*[2]

[2] $[t,t']$ is not required to be strongly-atomic.

/* $T_{\mathcal{J}} = \{t_1, t_2, \cdots, t_N\}$ */
1: **foreach** (strongly-atomic interval $[t_i, t_j]$)
2: $g_{i,j} := E(\mathcal{S}_{\rm opt}^{\mathcal{J}_{[t_i,t_j]}})$
3: **end foreach**
4: $\mathbf{V} := \{v_1, v_2, \cdots, v_N\}$
5: $\mathbf{E} := \{(v_i, v_j) \mid [t_i, t_j]$ is strongly-atomic$\}$
6: **foreach** $((v_i, v_j) \in \mathbf{E})$ $w((v_i, v_j)) := g_{i,j}$ **end foreach** /* weight of edges */
7: Find the shortest path from v_1 to v_N in $\mathbf{G} = (\mathbf{V}, \mathbf{E})$. /* Note that $\mathbf{G}$ is acyclic. */
/* The shortest path = $\langle v_{q_1}, v_{q_2}, \cdots v_{q_k} \rangle$ $(v_{q_1} = v_1, v_{q_k} = v_N)$ */
8: return $\oplus_{j=1}^{k-1} \mathcal{S}_{\rm opt}^{\mathcal{J}_{[t_{q_j}, t_{q_{j+1}}]}}$

Figure 10. An exponential-time optimal algorithm based on strongly-atomic intervals.

For the job set in Figure 3 not only $[r_3, r_2], [r_2, d_2]$ (Figure 3.(b)) and $[r_3, r_1]$ (Figure 3.(c)) but also $[r_1, d_2]$ (Figure 3.(c)) and $[r_3, d_3]$ (Figure 3.(d)) are strongly-atomic. Note that the intervals $[r_1, d_2]$ and $[r_3, d_3]$ are not covered by the previous definition in Section 5.1.1. Furthermore, (r_3, r_2, d_2) (Figure 3.(b)), (r_3, r_1, d_2) (Figure 3.(c)) and (r_3, d_3) (Figure 3.(d)) are strongly-blocking tuples.

Note that for an interval $[t, t']$, $\mathbf{I}_{\mathcal{J}_{[t,t']}} \subseteq [t, t']$ since $t \le r_J < d_J \le t'$ for all $J \in \mathcal{J}_{[t,t']}$. Therefore, for a strongly-blocking tuple $\mathbf{b} = (b_1, b_2, \cdots, b_k)$, $\mathbf{I}_{\mathcal{J}_{[b_1,b_2]}}, \mathbf{I}_{\mathcal{J}_{[b_2,b_3]}}, \cdots, \mathbf{I}_{\mathcal{J}_{[b_{k-1},b_k]}}$ are disjoint. Now, we prove that a job set can be partitioned by strongly-blocking tuples as with the job set in Figure 6 so that the formulation described in Section 5.1.1 can be extended to cover arbitrary job sets.

Lemma 13 *Given a job set $\mathcal{J}$ and an essential $|\mathcal{J}|$-tuple $\mathbf{f}$, $\mathcal{J}^{\mathbf{f}} \equiv \cup_{j=1}^{k-1} \mathcal{J}_j$ where $\mathbf{b}_{\sigma_{\mathbf{f}}} = (b_1, b_2, \cdots, b_k)$ and $\mathcal{J}_j$ is an EDF-equivalent job set of $\mathcal{J}_{[b_j, b_{j+1}]}$ for all $1 \le j < k$.*

Proof. Let $\mathcal{J}^{\mathbf{f}} = \{J'_1, J'_2, \cdots, J'_{|\mathcal{J}|}\}$ and let $\mathcal{J}_j = \{J'_l \in \mathcal{J}^{\mathbf{f}} \mid r_{J'_l} (= r_{J_l}) \in [b_j, b_{j+1})\}$. Then, $\{\mathcal{J}_1, \mathcal{J}_2, \cdots, \mathcal{J}_{k-1}\}$ forms a partition of $\mathcal{J}^{\mathbf{f}}$, i.e.,

$$\mathcal{J}^{\mathbf{f}} \equiv \cup_{j=1}^{k-1} \mathcal{J}_j \quad \text{and} \quad \mathcal{J}_j \cap \mathcal{J}_{j'} = \emptyset \ \text{ for all } 1 \le j \ne j' < k \, .$$

Thus, it suffices to show that $\mathcal{J}_j$ is an EDF-equivalent job set of $\mathcal{J}_{[b_j, b_{j+1}]}$ for all $1 \le j < k$. Let $i_j = \max\{\, i \mid f_{\sigma^{-1}(i)} = b_j\}$ for all $1 \le j \le k$, and suppose that $d_{J'_l} > b_{j+1}$ for a job $J'_l \in \mathcal{J}_j$. Then, we have $\sigma(l) > i_{j+1}$ since

$$f_{\sigma^{-1}(\sigma(l))} = f_l = d_{J'_l} > b_{j+1} = f_{\sigma^{-1}(i_{j+1})} \, .$$

From line 8 of the algorithm in Figure 9, we have

$$b_{j+1} = f_{\sigma^{-1}(i_{j+1})} \le \min\{r_{J_{\sigma^{-1}(k)}} \mid i_{j+1} < k \le |\mathcal{J}|\} \le r_{J_{\sigma^{-1}(k)}}{}^{|k=\sigma(l)\,(>i_{j+1})} = r_{J_l} \, ,$$

which contradicts $r_{J'_l} (= r_{J_l}) \in [b_j, b_{j+1})$. Therefore, $d_{J'_l} \in [b_j, b_{j+1}]$ for all $J'_l \in \mathcal{J}_j$. Furthermore, $\mathcal{J}_j$ follows the EDF priority since it is a subset of the EDF job set $\mathcal{J}^{\mathbf{f}}$.

It remains to show that $|\mathcal{J}_j| = |\mathcal{J}_{[b_j, b_{j+1}]}|$ and there is a bijective function $\alpha : \mathcal{J}_{[b_j, b_{j+1}]} \Rightarrow \mathcal{J}_j$ such that

$$\forall J' \in \mathcal{J}_{[b_j, b_{j+1}]} \, , \quad p_{J'} = p_{\alpha(J')} \, , \ c_{J'} = c_{\alpha(J')} \ \text{and} \ r_{J'} = r_{\alpha(J')} \, . \tag{11}$$

For the former, we have

$$|\mathcal{J}_j| = |\{J' \in \mathcal{J}^{\mathbf{f}} \mid r_{J'} \in [b_j, b_{j+1})\}| = |\{J \in \mathcal{J} \mid r_J \in [b_j, b_{j+1})\}| = |\mathcal{J}_{[b_j, b_{j+1}]}| \, .$$

For the latter, we define α such that $\alpha(J') = J''$ iff $p_{J'} = p_{J''}$. Then, it is clear that α is a bijective function and (11) holds. □

Lemma 14 *Let $\mathcal{S}(t) = \oplus_{j=1}^{h-1} \mathcal{S}_{\rm opt}^{\mathcal{J}_{[t_j, t_{j+1}]}}$ for $\min R_{\mathcal{J}} = t_1 < t_2 < \cdots < t_h = \max D_{\mathcal{J}}$ $(t_j \in T_{\mathcal{J}})$. Then, $\mathcal{S}$ is a feasible voltage schedule of $\mathcal{J}$. Furthermore,*

$$E(\mathcal{S}) = \sum_{j=1}^{h-1} E(\mathcal{S}_{\rm opt}^{\mathcal{J}_{[t_j, t_{j+1}]}}) \ge E(\mathcal{S}_{\rm opt}^{\mathcal{J}}) \, .$$

1: $f_{\sigma^{-1}(|\mathcal{J}|)} := d_{J_{\sigma^{-1}(|\mathcal{J}|)}}$
2: $\mathbf{b}^{\mathrm{w}}_{\sigma} := (d_{J_{\sigma^{-1}(|\mathcal{J}|)}})$
3: **for** ($i := |\mathcal{J}|-1$ to 1)
4: **let** $\mathcal{J}^{\mathrm{H}}$ be $\{J_{\sigma^{-1}(k)} \mid i < k \le |\mathcal{J}| \wedge \sigma^{-1}(k) < \sigma^{-1}(i)\}$
5: **if** ($r_{J_{\sigma^{-1}(i)}} \ge \min(\{r_J \mid J \in \mathcal{J}^{\mathrm{H}}\} \cup \{f_{\sigma^{-1}(i+1)}\})$) **return** FALSE
6: **else** $f_{\sigma^{-1}(i)} := \min(\{f_{\sigma^{-1}(i+1)}, d_{J_{\sigma^{-1}(i)}}\} \cup \{r_J \mid J \in \mathcal{J}^{\mathrm{H}}\})$
7: **end if**
8: **if** ($f_{\sigma^{-1}(i)} \le \min\{r_J \mid r \in \mathcal{J}^{\mathrm{H}}\}$) append $f_{\sigma^{-1}(i)}$ onto the head of $\mathbf{b}^{\mathrm{w}}_{\sigma}$
9: **end if**
10: **end for**
11: append min $R_{\mathcal{J}}$ onto the head of $\mathbf{b}^{\mathrm{w}}_{\sigma}$

Figure 11. The algorithm to build a weakly-blocking tuple from a $|\mathcal{J}|$-permutation.

Proof. Let $u_{[t_0,t_0']}(t)$ be defined by

$$u_{[t_0,t_0']}(t) = \begin{cases} 1 & t_0 \le t \le t_0' \,, \\ 0 & \text{otherwise} \,, \end{cases}$$

Since $\mathbf{I}_{[t_j,t_{j+1}]} \subseteq [t_j,t_{j+1}]$, $\mathcal{S}$ is feasible if $\mathcal{S}(t) \cdot u_{[t_j,t_{j+1}]}(t)$ is a feasible schedule of $\mathcal{J}_{[t_j,t_{j+1}]}$ for all $1 \le j < h$. By definition, $\mathcal{S}(t) \cdot u_{[t_j,t_{j+1}]}(t) = \mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[t_j,t_{j+1}]}}$ is a feasible schedule of $\mathcal{J}_{[t_j,t_{j+1}]}$ for all $1 \le j < h$. $E(\mathcal{S}) = \sum_{j=1}^{h-1} E(\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[t_j,t_{j+1}]}})$ holds trivially from $\mathbf{I}_{[t_j,t_{j+1}]} \subseteq [t_j,t_{j+1}]$. Finally, since $\mathcal{S}$ is feasible, $E(\mathcal{S}) \ge E(\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}})$. □

The following lemma implies how an energy-optimal voltage scheduling problem can be partitioned into subproblems.

Lemma 15 *Let*

$$E_1 = \min\{ \sum_{j=1}^{k-1} E(\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[b_j,b_{j+1}]}}) \mid (b_1,b_2,\cdots,b_k) \text{ is a strongly-blocking tuple.} \} \,,$$

$$E_2 = \min\{ \sum_{j=1}^{h-1} E(\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[t_j,t_{j+1}]}}) \mid \min R_{\mathcal{J}} = t_1 < t_2 < \cdots < t_h = \max D_{\mathcal{J}} \,,\, t_j \in T_{\mathcal{J}} \} \quad \text{and}$$

$$E_3 = \min\{ \sum_{j=1}^{h-1} E(\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[t_j,t_{j+1}]}}) \mid [t_j,t_{j+1}] \text{ is a sub-interval of a strongly-atomic interval for all } 1 \le j < h \} \,.$$

Then, $E(\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}}) = E_1 = E_2 = E_3$.

Proof. Let

$$\mathbf{S}_1 = \{ \oplus_{j=1}^{k-1} \mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[b_j,b_{j+1}]}} \mid (b_1,b_2,\cdots,b_k) \text{ is a strongly-blocking tuple.} \}$$

and define $\mathbf{S}_2$ and $\mathbf{S}_3$ similarly. Then, from Lemma 14, $E_i = \min\{E(\mathcal{S}) \mid \mathcal{S} \in \mathbf{S}_i\}$ for $i = 1,2,3$. By definition, $\mathbf{S}_1 \subseteq \mathbf{S}_3 \subseteq \mathbf{S}_2$ and consequently $E_2 \le E_3 \le E_1$. Furthermore, $E(\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}}) \le E_2$ from Lemma 14. From Theorem 5 and Lemma 13, $\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}} \in \mathbf{S}_1$. Thus, we have $E(\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}}) \ge E_1$, which implies $E(\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}}) = E_1 = E_2 = E_3$. □

Figure 10 shows an optimal algorithm which is based on strongly-atomic intervals. From Lemma 15, is is obvious that the algorithm in Figure 10 always computes an optimal voltage schedule. The algorithm may work efficiently for some job sets (e.g., the job set in Figure 6). But, the running time may not be bounded by a polynomial function; For the job set in Figure 7, there are only one strongly-atomic interval $[r_4,d_4]$ and the algorithm cannot but enumerate all the essential job sets. Furthermore, the algorithm does not have a structure suitable to be transformed into an FPTAS. So, we consider another optimal algorithm based on *weakly-atomic* intervals, *weakly-bounding* tuples, and the background workload. First, we formally define the terms based on the algorithm in Figure 11, which is identical to the algorithm in Figure 9 except for the boxed code segment (line 8).

Definition 16 *Given a valid $|\mathcal{J}|$-permutation σ, the tuple $\mathbf{b}^{\mathrm{w}}_{\sigma}$ built by the algorithm in Figure 11 is called a* weakly-blocking tuple. *An interval $[t,t']$ is* weakly-atomic *if there is a weakly-blocking tuple $\mathbf{b}^{\mathrm{w}} = (b_1,b_2,\cdots,b_k)$ such that $[t,t'] = [b_i,b_{i+1}]$ for some $1 \le i < k$. Furthermore, the job set $\mathcal{J}_{[t,t']^{\mathrm{w}}}$ defined by*

$$\mathcal{J}_{[t,t']^{\mathrm{w}}} = \{J' \mid J \in \mathcal{J} \,,\, r_J \in [t,t') \wedge (\exists J_{\mathrm{h}} \in \mathcal{J}, p_{J_{\mathrm{h}}} < p_J \wedge r_{J_{\mathrm{h}}} = t' \wedge d_J \in [r_{J_{\mathrm{h}}}, d_{J_{\mathrm{h}}}))\} \text{ where}$$
$$r_{J'} = r_J, c_{J'} = c_J, p_{J'} = p_J \text{ and } d_{J'} = \min\{d_J,t'\} \,.$$

is said to be weakly-induced *by an interval* $[t,t']$.
Furthermore, given a weakly-blocking tuple $\mathbf{b}^{\mathrm{w}} = (b_1, b_2, \cdots, b_k)$ *and the corresponding EDF-equivalent job set* $\mathcal{J}'$*, any job in* $\mathcal{J}' - \cup_{j=1}^{k-1} \mathcal{J}_{[b_j, b_{j+1}]^{\mathrm{w}}}$ *is called a* background job *with respect to the weakly-blocking tuple* $\mathbf{b}^{\mathrm{w}}$*. The workload of background jobs are called* background workload.

Note that for an interval $[t,t']$, $\mathbf{I}_{\mathcal{J}_{[t,t']^{\mathrm{w}}}} \subseteq \mathbf{I}_{\mathcal{J}_{[t,t']}} \subseteq [t,t']$ since $\mathcal{J}_{[t,t']^{\mathrm{w}}} \subseteq \mathcal{J}_{[t,t']}$. Therefore, for a weakly-blocking tuple $\mathbf{b}^{\mathrm{w}} = (b_1, b_2, \cdots, b_k)$, $\mathbf{I}_{\mathcal{J}_{[b_1,b_2]^{\mathrm{w}}}}, \mathbf{I}_{\mathcal{J}_{[b_2,b_3]^{\mathrm{w}}}}, \cdots, \mathbf{I}_{\mathcal{J}_{[b_{k-1},b_k]^{\mathrm{w}}}}$ are disjoint.

Lemma 17 *Given a weakly-blocking tuple* $\mathbf{b}^{\mathrm{w}}$*, let* $\mathcal{J}^{\mathcal{B}}_{\mathbf{b}^{\mathrm{w}}}$ *represent the set of background jobs with respect to* $\mathbf{b}^{\mathrm{w}}$*. Then,* $\mathcal{J}^{\mathcal{B}}_{\mathbf{b}^{\mathrm{w}}_1} \equiv \mathcal{J}^{\mathcal{B}}_{\mathbf{b}^{\mathrm{w}}_2}$ *for any weakly-blocking tuples* $\mathbf{b}^{\mathrm{w}}_1$ *and* $\mathbf{b}^{\mathrm{w}}_2$*.*

Proof. Let $\mathbf{b}^{\mathrm{w}}_1 = (b_1, b_2, \cdots, b_k)$ and $\mathbf{b}^{\mathrm{w}}_2 = (b'_1, b'_2, \cdots, b'_{k'})$. Assume that $J \in \mathcal{J}^{\mathcal{B}}_{\mathbf{b}^{\mathrm{w}}_1}$ and $r_J \in [b_j, b_{j+1})$. From the definition of a background job, we have

$$\exists k > j+1\ , \quad d_J \geq b_{j+1} \ \text{ and } \ p_J > \max\{p_{J'} | J' \in \cup_{l=j+1}^{k-1} \mathcal{J}_{[b_l, b_{l+1}]^{\mathrm{w}}}\}\ . \tag{12}$$

Suppose that $J \notin \mathcal{J}^{\mathcal{B}}_{\mathbf{b}^{\mathrm{w}}_2}$. From (12), we have

$$[b_{j+1}, b_{j+2}] \subseteq (b'_{j'}, b'_{j'+1}] \ \text{ for } \ r_J \in [b'_{j'}, b'_{j'+1})\ ,$$

a contradiction. So, $\mathcal{J}^{\mathcal{B}}_{\mathbf{b}^{\mathrm{w}}_1} \subseteq \mathcal{J}^{\mathcal{B}}_{\mathbf{b}^{\mathrm{w}}_2}$. Similarly, we have $\mathcal{J}^{\mathcal{B}}_{\mathbf{b}^{\mathrm{w}}_2} \subseteq \mathcal{J}^{\mathcal{B}}_{\mathbf{b}^{\mathrm{w}}_1}$. □

Lemma 17 states that we can specify background jobs irrespective of weakly-blocking tuples. For the rest of this paper, we use $\mathcal{J}^{\mathcal{B}}$ to represent the set of background jobs.

Lemma 18 *Given a job set* $\mathcal{J}$ *and an essential* $|\mathcal{J}|$*-tuple* $\mathbf{f}$*, let* $\mathbf{b}^{\mathrm{w}}_{\sigma_{\mathbf{f}}} = (b_1, b_2, \cdots, b_k)$*. Then, for any weakly-atomic interval* $[b_j, b_{j+1}]$ $(1 \leq j < k)$ *and a background job* J*, we have the following assuming jobs are executed under* $\mathcal{S}^{\mathcal{J}}_{\mathrm{opt}}$*.*
(a) $d_J \in [b_j, b_{j+1})$*: J completes its execution by* b_j*.*
(b) $r_J \in [b_j, b_{j+1})$*: J completes its execution by* b_{j+1}*.*
(c) $[b_j, b_{j+1}] \subseteq [r_J, d_J]$ *executes its partial workload at constant speed.*
Furthermore, for any interval $[t,t'] \subseteq [b_j, b_{j+1}]$*,* $\mathcal{J}_{[t,t']^{\mathrm{w}}}$ *is an EDF job set.*

Proof. Case (a) and Case (b) are obvious from the construction of the weakly-blocking tuple $\mathbf{b}^{\mathrm{w}}_{\sigma_{\mathbf{f}}}$. Case (c) follows from Lemma 9. Finally, suppose that $\mathcal{J}_{[t,t']^{\mathrm{w}}}$ is not an EDF job set. Then, we have

$$\exists J, J' \in \mathcal{J}_{[t,t']^{\mathrm{w}}} \text{ s.t. } \ p_J > p_{J'}\ , \quad d_J \in (r_{J'}, d_{J'})\ ,$$

and the algorithm in Figure 11 selects $r_{J'}$ $(\in (b_j, b_{j+1}))$ as an element of $\mathbf{b}^{\mathrm{w}}_{\sigma_{\mathbf{f}}}$, a contradiction. □

From Lemma 18, we characterize the optimal schedule in terms of weakly-atomic intervals, weakly-blocking tuples and background workload.

Lemma 19 *Given a job set* $\mathcal{J}$ *and an essential* $|\mathcal{J}|$*-tuple* $\mathbf{f}$*,*

$$\mathcal{S}^{\mathcal{J}}_{\mathrm{opt}} \equiv \oplus_{j=1}^{k-1} \mathcal{S}^{\mathcal{J}_j}_{\mathrm{opt}} \tag{13}$$

where $\mathbf{b}^{\mathrm{w}}_{\sigma_{\mathbf{f}}} = (b_1, b_2, \cdots, b_k)$ *and* $\mathcal{J}_j = \mathcal{J}_{[b_j, b_{j+1}]} \cup \{J^{\mathrm{b}}_j\}$ *such that*

$$r_{J^{\mathrm{b}}_j} = b_j\ , \ d_{J^{\mathrm{b}}_j} = b_{j+1}\ , \ p_{J^{\mathrm{b}}_j} = \max\{p_J | J \in \mathcal{J}_{[b_j, b_{j+1}]}\} + 1 \ \textit{and} \ c_{J^{\mathrm{b}}_j} = c^{\mathrm{b}}_j \ \textit{for some} \ c^{\mathrm{b}}_j \geq 0\ .$$

Proof. From Lemma 18, we have

$$\begin{aligned} &\{job(\mathcal{J}, \mathcal{S}^{\mathcal{J}}_{\mathrm{opt}}(t), t) \mid t \in [b_j, b_{j+1})\} \ \equiv \ \mathcal{J}_{[b_j, b_{j+1}]^{\mathrm{w}}} \cup \mathcal{J}' \cup \mathcal{J}'' \ \text{ where} \\ &\mathcal{J}' \ = \ \{J' \in \mathcal{J}^{\mathcal{B}} | r_{J'} \in [b_j, b_{j+1})\} \quad \text{and} \quad \mathcal{J}'' \ = \ \{J' \in \mathcal{J}^{\mathcal{B}} | [b_j, b_{j+1}] \subseteq [r_{J'}, d_{J'}]\}\ . \end{aligned}$$

From Case (b) of Lemma 18, $\mathcal{J}_{[b_j, b_{j+1}]^{\mathrm{w}}} \cup \mathcal{J}' \equiv \mathcal{J}_{[b_j, b_{j+1}]}$, and from Case (c), $\mathcal{J}'' = \{J_{\mathrm{b}}\}$. So, we have

$$\mathcal{S}^{\mathcal{J}}_{\mathrm{opt}}(t) \cdot u_{[b_j, b_{j+1}]}(t) \ \equiv \ \mathcal{S}^{\mathcal{J}_j}_{\mathrm{opt}} \quad \text{for all } \ 1 \leq j < k\ ,$$

which is equivalent to (13). □

From Lemma 19, the voltage scheduling problem is reduced to the problem of finding a weakly-blocking tuple $\mathbf{b}^{\mathrm{w}} = (b_1, b_2, \cdots, b_k)$ and the amount of background workload $c^{\mathrm{B}}_{[b_j, b_{j+1}]}$ for each weakly-atomic interval $[b_j, b_{j+1}]$. To find the *background speed* $s^{\mathrm{B}}_{[b_j, b_{j+1}]}$ instead of the amount of background workload makes it possible to exploit Lemma 9.

Lemma 20 *Given a weakly-atomic interval $[t_1, t_2]$, let $\mathcal{J}' = \mathcal{J}_{[t_1,t_2]} \cup \{J^{\mathrm{b}}\}$ where*

$$r_{J^{\mathrm{b}}} = t_1 \ , \ d_{J^{\mathrm{b}}} = t_2 \ , \ p_{J^{\mathrm{b}}} = \max\{p_J | J \in \mathcal{J}_{[t_1,t_2]}\} + 1 \ \textit{and} \ c_{J^{\mathrm{b}}} = c^{\mathrm{B}}_{[t_1,t_2]} \ (> 0) \ ,$$

and let $s^{\mathrm{B}}_{[t_1,t_2]}$ be the constant speed of J^{b} under $\mathcal{S}^{\mathcal{J}'}_{\mathrm{opt}}$. Then,

$$\mathcal{S}^{\mathcal{J}'}_{\mathrm{opt}}(t) = \begin{cases} \mathcal{S}^{\mathcal{J}_{[t_1,t_2]}}_{\mathrm{opt}}(t) & t \ \ s.t. \ \ \mathcal{S}^{\mathcal{J}_{[t_1,t_2]}}_{\mathrm{opt}}(t) > s^{\mathrm{B}}_{[t_1,t_2]} \ , \\ s^{\mathrm{B}}_{[t_1,t_2]} & t \ \ s.t. \ \ \mathcal{S}^{\mathcal{J}_{[t_1,t_2]}}_{\mathrm{opt}}(t) \le s^{\mathrm{B}}_{[t_1,t_2]} \ . \end{cases}$$

Furthermore, $s^{\mathrm{B}}_{[t_1,t_2]}$ strictly increases as $c^{\mathrm{B}}_{[t_1,t_2]}$ increases, and vice versa.

Proof. From Lemmas 18 and 19, both $\mathcal{J}_{[t_1,t_2]}$ and $\mathcal{J}'$ follow the EDF priority and their optimal voltage schedules are obtained by Yao's algorithm [22]. For an interval $[t_1', t_2'] \subset [t_1, t_2]$ such that $\mathcal{S}^{\mathcal{J}_{[t_1',t_2']}}_{\mathrm{opt}}(t) > s^{\mathrm{B}}_{[t_1',t_2']}$, Yao's algorithm selects the same speed for $\mathcal{S}^{\mathcal{J}'}_{\mathrm{opt}}(t)$. For the other intervals, $\mathcal{S}^{\mathcal{J}'}_{\mathrm{opt}}(t) = s_{J^{\mathrm{b}}}$ since $[t_1, t_2] \subseteq [r_{J^{\mathrm{b}}}, d_{J^{\mathrm{b}}}]$.

Because $W(\mathcal{S}^{\mathcal{J}'}_{\mathrm{opt}}, [t_1, t_2])$ strictly increases as $s^{\mathrm{B}}_{[t_1,t_2]}$ increases and $c^{\mathrm{B}}_{[t_1,t_2]} = W(\mathcal{S}^{\mathcal{J}'}_{\mathrm{opt}}, [t_1, t_2]) - W(\mathcal{S}^{\mathcal{J}_{[t_1,t_2]}}_{\mathrm{opt}}, [t_1, t_2])$, $c^{\mathrm{B}}_{[t_1,t_2]}$ increases as $s^{\mathrm{B}}_{[t_1,t_2]}$ increases. Hence, it follows that $s^{\mathrm{B}}_{[t_1,t_2]}$ increases as $c^{\mathrm{B}}_{[t_1,t_2]}$ increases (and vice versa). □

Definition 21 *Given a job set $\mathcal{J}$ and background workload c, the job set $\mathcal{J}$ with background workload c is defined as*

$$\mathcal{J}[c] \stackrel{def}{=} \mathcal{J} \cup \{J^{\mathrm{b}}\} \ \ \textit{where} \ \ r_{J^{\mathrm{b}}} = R_{\mathcal{J}}, \ d_{J^{\mathrm{b}}} = D_{\mathcal{J}}, \ p_{J^{\mathrm{b}}} = \max\{p_J | J \in \mathcal{J}\} + 1 \ \textit{and} \ c_{J^{\mathrm{b}}} = c \ .$$

Furthermore, given a job set $\mathcal{J}[c]$, the constant speed of background workload under $\mathcal{S}^{\mathcal{J}[c]}_{\mathrm{opt}}$ is called a background speed *of $\mathcal{J}[c]$ and is denoted by $BS(\mathcal{J}, c)$.*

The following lemma is an extension of Lemma 20 for arbitrary intervals.

Lemma 22 *Given a job set $\mathcal{J}[c]$*

$$\begin{aligned} &\mathcal{S}^{\mathcal{J}[c]}_{\mathrm{opt}} \equiv \oplus_{j=1}^{k-1} \mathcal{S}^{\mathcal{J}_{[b_j,b_{j+1}]}[c_j]}_{\mathrm{opt}} \quad \textit{for} \ \ b_1, \cdots, b_k \in \mathcal{T}_{\mathcal{J}}, \ b_1 < \cdots < b_k \quad \textit{such that} \\ &c = \textstyle\sum_{j=1}^{k-1} c_j \quad \textit{and} \quad BS(\mathcal{J}_{[b_j,b_{j+1}]}, c_j) = BS(\mathcal{J}_{[b_{j'},b_{j'+1}]}, c_{j'}) \quad \textit{for all} \ \ 1 \le j \ne j' < c_j \ . \end{aligned}$$

Proof. Directly from Lemmas 19 and 9. □

Along with Lemma 22, the following lemma implies how the problem can be reduced to a dynamic programming formulation.

Lemma 23 *Given $t_i, t_j, t_m \in \mathcal{T}_{\mathcal{J}}$ where $t_i < t_m \le t_j$, let*

$$\begin{aligned} \mathcal{J}^{\mathrm{B}}_{[t_i,t_j]^{\mathrm{w}}} &= \mathcal{J}_{[t_i,t_j]^{\mathrm{w}}} \cup \{J \in \mathcal{J}^{\mathcal{B}} \mid [r_J, d_J] \subseteq [t_i, t_j]\} \quad \textit{and} \\ c^{\mathrm{B}}_{[t_i,t_m]} &= C(\{J \in \mathcal{J}^{\mathcal{B}} \mid r_J \in [t_i, t_m) \wedge d_J \in [t_m, t_j] \ \}) \ , \end{aligned}$$

and let $\mathcal{S}^{[t_i,t_j]}_{\mathrm{opt}}$ represent $\mathcal{S}^{\mathcal{J}^{\mathrm{B}}_{[t_i,t_j]^{\mathrm{w}}}}_{\mathrm{opt}}$. Then,

$$\mathcal{S}^{[t_i,t_j]}_{\mathrm{opt}} \ \in \ \{\mathcal{S}^{\mathcal{J}_{[t_i,t_m]^{\mathrm{w}}}[c^{\mathrm{B}}_{[t_i,t_m]}]}_{\mathrm{opt}} \oplus \mathcal{S}^{[t_m,t_j]}_{\mathrm{opt}} \mid \mathcal{S}^{\mathcal{J}_{[t_i,t_m]^{\mathrm{w}}}[c^{\mathrm{B}}_{[t_i,t_m]}]}_{\mathrm{opt}} \ \textit{is feasible for} \ \ \mathcal{J}^{\mathrm{B}}_{[t_i,t_m]^{\mathrm{w}}}\} \ .$$

procedure OPTIMAL_VOLTAGE_SCHEDULE ($\mathcal{J}$)

/* $T_{\mathcal{J}} = \{t_1, t_2, \cdots, t_N\}$, $S_{\rm C} := \{s_1, s_2, \cdots, s_n\}$ */

foreach ($s \in S_{\rm C}$)
  $\mathbf{V} := \{v_1, v_2, \cdots, v_N\}$
  $\mathbf{E} := \{(v_i, v_j) \mid [t_i, t_j]$ is weakly-atomic$\}$
  foreach $((v_i, v_j) \in \mathbf{E})$
    $w((v_i, v_j)) := W(\max\{\mathcal{S}_{\rm opt}^{\mathcal{J}_{[t_i,t_j]}}(t), s\}, [t_i, t_j]) - W(\mathcal{S}_{\rm opt}^{\mathcal{J}_{[t_i,t_j]}}(t), [t_i, t_j])$ /* weight of edges */
  end foreach
  Find longest paths between all pairs of vertices in $\mathbf{V}$. /* Note that G is acyclic. */
  foreach ($1 \le i < j \le N$ s.t. $[t_i, t_j]$ is a concatenation of weakly-atomic intervals)
    /* The longest path from v_i to v_j = $\langle v_{q_1}, v_{q_2}, \cdots v_{q_l} \rangle$
    $c :=$ the weight of the longest path from v_i to v_j.
    $E_{i,j}[c] := E(\oplus_{h=1}^{l-1} \max\{ \mathcal{S}_{\rm opt}^{\mathcal{J}_{[t_{q_j}, t_{q_{j+1}}]}}(t), s\}, [t_i, t_j])$
  end foreach
end foreach
for ($i := 1$ to $N-1$)
  for ($j := 1$ to $N-i$)
    $E_{j,j+i} := \infty^+$
    for ($k := j+1$ to $j+i$)
      $c_{j,j+i,k} := C(\{J \in \mathcal{J}^{\mathcal{B}} \mid r_J \in [t_j, t_k) \wedge d_J \in [t_k, t_{j+i}]\})$
      $E_{j,j+i,k} := E_{j,k}[c_{j,j+i,k}] + E_{k,j+i}$
      if ($E_{j,j+i} > E_{j,j+i,k}$ **and** $\mathcal{S}_{\rm opt}^{\mathcal{J}_{[t_j,t_k]^{\rm w}}[c_{j,j+i,k}]}$ is feasible for $\mathcal{J}_{[t_i,t_j]^{\rm w}} \cup \{J \in \mathcal{J}^{\mathcal{B}} | [r_J, d_J] \subseteq [t_i, t_j]\}$)
        $E_{j,j+i} := E_{j,j+i,k}$, $h := k$
      end if
    end for
    $\mathbf{b}_{j,j+i} := \{t_h\} \cup \mathbf{b}_{j,h} \cup \mathbf{b}_{h,j+i}$
  end for
end for

/* $E_{1,N} = E(\mathcal{S}_{\rm opt}^{\mathcal{J}})$ and $\mathcal{S}_{\rm opt}^{\mathcal{J}} \equiv \mathcal{S}_{\rm opt}^{\cup_{h=1}^{l-1} \mathcal{J}_{[b_h, b_{h+1}]^{\rm w}} \cup \mathcal{J}^{\mathcal{B}}}$ where $\mathbf{b}_{1,N} = (b_1, b_2, \cdots, b_l)$ */

$\mathcal{J}_{\rm opt} := \cup_{h=1}^{l-1} \mathcal{J}_{[b_h, b_{h+1}]^{\rm w}} \cup \mathcal{J}^{\mathcal{B}}$ where $\mathbf{b}_{1,N} = (b_1, b_2, \cdots, b_l)$

return $\mathcal{S}_{\rm opt}^{\mathcal{J}_{\rm opt}}$ /* $\mathcal{J}_{\rm opt}$ is an EDF job set. So, $\mathcal{S}_{\rm opt}^{\mathcal{J}_{\rm opt}}$ can be directly computed by Yao's algorithm */

end procedure

Figure 12. An exponential-time optimal algorithm based on weakly-atomic intervals.

Proof. If all the jobs in $\{J \in \mathcal{J}^{\mathcal{B}} \mid [r_J, d_J] \subseteq [t_i, t_j]\}$ runs at the same speed under $\mathcal{S}_{\rm opt}^{[t_i,t_j]}$, $\mathcal{S}_{\rm opt}^{[t_i,t_j]} \equiv \mathcal{S}_{\rm opt}^{\mathcal{J}_{[t_i,t_j]^{\rm w}}[c^{\rm B}_{[t_i,t_j]}]}$. Otherwise, there must exist $t_m \in \mathcal{T}_{\{J \in \mathcal{J}^{\mathcal{B}} \mid [r_J, d_J] \subseteq [t_i, t_j]\}}$ $(\subseteq \mathcal{T}_{\mathcal{J}})$ such that all the jobs in $\{J \in \mathcal{J}^{\mathcal{B}} \mid r_J \in [t_i, t_m) \wedge d_J \in [t_m, t_j]\}$ finish their executions by t_m with the same constant speed and all the jobs in $\{J \in \mathcal{J}^{\mathcal{B}} \mid [r_J, d_J] \subseteq [t_m, t_l]\}$ are not executed before t_m under $\mathcal{S}_{\rm opt}^{[t_i,t_j]}$. Therefore, we have $\mathcal{S}_{\rm opt}^{[t_i,t_j]} \equiv \mathcal{S}_{\rm opt}^{\mathcal{J}_{[t_i,t_m]^{\rm w}}[c^{\rm B}_{[t_i,t_m]}]} \oplus \mathcal{S}_{\rm opt}^{[t_m,t_j]}$ where $\mathcal{S}_{\rm opt}^{\mathcal{J}_{[t_i,t_m]^{\rm w}}[c^{\rm B}_{[t_i,t_m]}]}$ is feasible for $\mathcal{J}^{\rm B}_{[t_i,t_m]^{\rm w}}$. □

Corollary 24 *Let $E_{\rm opt}^{[t_i,t_j]}$ denote $E(\mathcal{S}_{\rm opt}^{[t_i,t_j]})$ where $\mathcal{S}_{\rm opt}^{[t_i,t_j]}$ is defined as in Lemma 23. Then,*

$$E_{\rm opt}^{[t_i,t_j]} = \min\left(\{E(\mathcal{S}_{\rm opt}^{\mathcal{J}_{[t_i,t_m]^{\rm w}}[c^{\rm B}_{[t_i,t_m]}]}) + E_{\rm opt}^{[t_m,t_j]} \mid t_m \in \mathcal{T}_{\mathcal{J}},\ t_i < t_m < t_k,\ \mathcal{S}_{\rm opt}^{\mathcal{J}_{[t_i,t_m]^{\rm w}}[c^{\rm B}_{[t_i,t_m]}]} \text{ is feasible for } \mathcal{J}^{\rm B}_{[t_i,t_m]^{\rm w}}\} \right) .$$

Based on weakly-atomic interval, we construct another optimal voltage scheduling algorithm. Figure 12 shows the optimal algorithm which is based on the dynamic programming formulated by weakly-atomic intervals. The algorithm identifies weakly-atomic intervals and computes the optimal schedule for the weakly-atomic interval. (Note that jobs in a weakly-atomic interval follow the EDF priority assignment.) In computing the optimal schedule for a weakly-atomic interval, we consider the background workload, that is, the algorithm computes the optimal schedule for each candidate background speed in $S_{\rm C}$. Given a job set $\mathcal{J}$, the algorithm first computes the set $S_{\rm C}$ of candidates for the speed of background workload. For the optimal algorithm, the set $S_{\rm C}$ is set to be

$$S_{\rm C} = \{\frac{C(\mathcal{J}')}{\sum_{i=0}^{k-1}(t_{p_{2i+2}} - t_{p_{2i+1}})} \mid \mathcal{J}' \subset \mathcal{J},\ t_1 < t_2 < \cdots < t_{p_{2k}},\ t_j \in T_{\mathcal{J}}\} .$$

It is obvious that the speed of the background workload in an optimal voltage schedule is included in $S_{\rm C}$. (In the FPTAS which will be presented in Section 5.2, the set $S_{\rm C}$ is selected such that the size of $S_{\rm C}$ is bounded

by a polynomial function.) Given the optimal schedules of weakly-atomic intervals, the algorithm searches the minimum sum of the energy values of the weakly-atomic intervals. The correctness of the algorithm directly follows from Lemma 23 and Corollary 24. The worst-case running time of the algorithm is not bounded by a polynomial function, but it can be easily transformed into an FPTAS.

5.2 Approximation Algorithm

First, we prove a miscellaneous property which is useful in bounding the error of our approximation algorithm.

Lemma 25 *Given a function* $P : \mathbf{R}^+ \Rightarrow \mathbf{R}^+$ *and a constant* $0 < \varepsilon < 1$, *if*

$$0 < x_1 < x_2 < \left(1 + \frac{\varepsilon \cdot \log 2}{\max\{\eta(x) | x > 0\}}\right) \cdot x_1 \quad \textit{where} \quad \eta(x) = \frac{P'(x)}{P(x)} \cdot x \ ,$$
$$\textit{then} \quad P(x_2) < (1+\varepsilon) \cdot P(x_1) \ .$$

Proof. From the condition, we have

$$\log x_2 - \log x_1 < \log\left(1 + \frac{\varepsilon \cdot \log 2}{\max\{\eta(x) | x > 0\}}\right) < \frac{\varepsilon \cdot \log 2}{\max\{\eta(x) | x > 0\}} \ . \tag{14}$$

Let $y_1 = \log x_1$ and $y_2 = \log x_2$. Then we have

$$\log P(x_2) - \log P(x_1) = \log P(e^{y_2}) - \log P(e^{y_1}) \leq (y_2 - y_1) \cdot \max\{ \frac{d(\log P(e^y))}{dy} \} \ .$$

From (14) and

$$\frac{d(\log P(e^y))}{dy} = \frac{P'(e^y)}{P(e^y)} \cdot e^y = \eta(e^y) \ ,$$

we have

$$\log P(x_2) - \log P(x_1) < \frac{\varepsilon \cdot \log 2}{\max\{\eta(x) | x > 0\}} \cdot \max\{ \eta(x) \mid x > 0\} = \varepsilon \cdot \log 2 \ .$$

It follows that

$$P(x_2) < e^{\varepsilon \cdot \log 2} \cdot P(x_1) < e^{\log(1+\varepsilon)} \cdot P(x_1) = (1+\varepsilon) \cdot P(x_1) \ .$$

□

For a power function $P(s) = \alpha \cdot s^n$, we have $\eta(s) = n$. In the following, we use ρ_P to denote $\log 2 / \max\{\eta(x) | x > 0\}$. From Lemma 25, we can construct an FPTAS as in Figure 13. The FPTAS is slightly different from the algorithm in Figure 12. To bring the running time down to polynomial, we use S'_{C} instead of S_{C}:

$$S'_{\mathrm{C}} = \{\min\{S_{\mathrm{C}}\} \cdot (1+\varepsilon \cdot \rho_P)^k \mid k = 0, 1, \cdots, l \text{ where } \min\{S_{\mathrm{C}}\} \cdot (1+\varepsilon \cdot \rho_P)^{l-1} < \max\{S_{\mathrm{C}}\} \leq \min\{S_{\mathrm{C}}\} \cdot (1+\varepsilon \cdot \rho_P)^l \} \ .$$

Theorem 26 APPROX_VOLTAGE_SCHEDULE *is a fully polynomial time approximation scheme for the voltage scheduling problem.*

Proof. Let s_1 and s_2 be elements of S'_{C} such that $s_2 = s_1 \cdot (1+\varepsilon \cdot \rho_P)$. Given a weakly-atomic interval $[t_i, t_j]$, we have for $t_i \leq t \leq t_j$:

$$\max\{\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[t_i,t_j]}}(t), s_2\} \leq (1+\varepsilon \cdot \rho_P) \cdot \max\{\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[t_i,t_j]}}(t), s_1\} \ .$$

Thus, from Lemma 25, we have for $t_i \leq t \leq t_j$

$$P(\max\{\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[t_i,t_j]}}(t), s_2\}) \leq (1+\varepsilon) \cdot P(\max\{\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[t_i,t_j]}}(t), s_1\}) \quad \text{, which implies}$$
$$E(\max\{\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[t_i,t_j]}}(t), s_2\}, [t_i, t_j]) \leq (1+\varepsilon) \cdot E(\max\{\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[t_i,t_j]}}(t), s_1\}, [t_i, t_j]) \ .$$

Let us compare $E_{j,k}[c']$ in line 21 of APPROX_VOLTAGE_SCHEDULE and $E_{j,k}[c_{j,j+i,k}]$ in line 18 of OPTIMAL_VOLTAGE_SCHEDULE. Let s' and s be the corresponding elements in S'_{C} and S_{C}, respectively. Then, from the definition of S'_{C}, we have $s' < (1+\varepsilon \cdot \rho_P) \cdot s$, which implies $E_{j,k}[c'] < (1+\varepsilon) \cdot E_{j,k}[c_{j,j+i,k}]$. Therefore, $E_{1,N} < (1+\varepsilon) \cdot E(\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}})$. Finally, since we have

$$|S'_{\mathrm{C}}| = 1 + \lceil \log_{1+\varepsilon \cdot \rho_P}(\max\{S_{\mathrm{C}}\} / \min\{S_{\mathrm{C}}\}) \rceil < 2 + \frac{\log(\max\{S_{\mathrm{C}}\} / \min\{S_{\mathrm{C}}\})}{\varepsilon \cdot \log(1+\rho_P)} \ , \tag{15}$$

the running time is bounded a polynomial function of $|\mathcal{J}|$ and $1/\varepsilon$. □

procedure APPROX_VOLTAGE_SCHEDULE ($\mathcal{J}, \varepsilon$)

/* $T_{\mathcal{J}} = \{t_1, t_2, \cdots, t_N\}$ */

/* $S'_{\mathrm{C}} = \{\min\{S_{\mathrm{C}}\} \cdot (1+\delta)^k | k = 0, 1, \cdots, \lceil \log_{1+\delta}(\max\{S_{\mathrm{C}}\}/\min\{S_{\mathrm{C}}\}) \rceil$ where $\delta = \varepsilon \cdot \rho_P$*/

Initialize $C_{i,j} := \{\}$ for $1 \le i < j \le N$.

foreach ($s \in S'_{\mathrm{C}}$)

$\mathbf{V} := \{v_1, v_2, \cdots, v_N\}$

$\mathbf{E} := \{(v_i, v_j) \mid [t_i, t_j]$ is weakly-atomic$\}$

foreach ($(v_i, v_j) \in \mathbf{E}$)

$w((v_i, v_j)) := W(\max\{\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[t_i,t_j]}}(t), s\}, [t_i, t_j]) - W(\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[t_i,t_j]}}(t), [t_i, t_j])$ /* weight of edges */

end foreach

Find longest paths between all pairs of vertices in $\mathbf{V}$. /* Note that G is acyclic. */

foreach ($1 \le i < j \le N$ s.t. $[t_i, t_j]$ is a concatenation of weakly-atomic intervals)

/* The longest path from v_i to v_j = $\langle v_{q_1}, v_{q_2}, \cdots v_{q_l} \rangle$

$c :=$ the weight of the longest path from v_i to v_j.

$E_{i,j}[c] := E(\oplus_{h=1}^{l-1} \max\{ \mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[t_{q_j}, t_{q_{j+1}}]}}(t), s\}, [t_i, t_j])$

$C_{i,j} := C_{i,j} \cup \{c\}$

end foreach

end foreach

for ($i := 1$ to $N-1$)

for ($j := 1$ to $N-i$)

$E_{j,j+i} := \infty^+$

for ($k := j+1$ to $j+i$)

$c_{j,j+i,k} := C(\{J \in \mathcal{J}^{\mathcal{B}} \mid r_J \in [t_j, t_k) \wedge d_J \in [t_k, t_{j+i}]\})$

$c' := \min\{c \in C_{j,k} | c \ge c_{j,j+i,k}\}$

$E_{j,j+i,k} := E_{j,k}[c'] + E_{k,j+i}$

if ($E_{j,j+i} > E_{j,j+i,k}$ **and**

$\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_{[t_j,t_k]^{\mathrm{w}}}[c_{j,j+i,k}]}$ is feasible for $\mathcal{J}_{[t_i,t_j]^{\mathrm{w}}} \cup \{J \in \mathcal{J}^{\mathcal{B}} | [r_J, d_J] \subseteq [t_i, t_j]\}$)

$E_{j,j+i} := E_{j,j+i,k}$, $h := k$

end if

end for

$\mathbf{b}_{j,j+i} := \{t_h\} \cup \mathbf{b}_{j,h} \cup \mathbf{b}_{h,j+i}$

end for

end for

/* $E_{1,N} < (1+\varepsilon) \cdot E(\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}})$ */

$\mathcal{J}_\varepsilon := \cup_{h=1}^{l-1} \mathcal{J}_{[b_h, b_{h+1}]^{\mathrm{w}}} \cup \mathcal{J}^{\mathcal{B}}$ where $\mathbf{b}_{1,N} = (b_1, b_2, \cdots, b_l)$

return $\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_\varepsilon}$ /* $\mathcal{J}_\varepsilon$ is an EDF job set. So, $\mathcal{S}_{\mathrm{opt}}^{\mathcal{J}_\varepsilon}$ can be directly computed by Yao's algorithm */

end procedure

Figure 13. The fully polynomial time approximation scheme.

6 Workload-Aware Optimal Off-Line Scheduling

The off-line voltage scheduling algorithm described in Section 5 is based on the assumption that the workload of each job is constant. Off-line algorithms can be applied to the case where each job runs at its worst-case execution time (WCET). However, the execution time of each job varies, sometimes by a large amount, which cannot be adequately handled by off-line scheduling alone. Therefore, rescheduling by an on-line scheduling algorithm is necessary during runtime. On-line scheduling is effective in leveraging the execution time variation, but it should not spend much computation time due to the runtime overhead.

Consequently, both the off-line scheduling and the on-line scheduling are needed in realizing the full potential of energy saving with sophisticated static analysis while exploiting the workload variation appropriately without incurring much runtime overhead. A naive approach to combine the off-line scheduling and the on-line scheduling is to decouple the off-line and on-line decisions. That is, the off-line scheduler assumes that each job runs at its WCET, and is unaware of either the workload variation or the behavior of the on-line scheduler. This policy makes the off-line scheduling problem easier, but the off-line schedule may lead to a poor energy efficiency once rescheduled by the on-line scheduler during runtime.

Given an off-line voltage schedule $\mathbf{A} = (a_1, a_2, \cdots, a_{|\mathcal{J}|})$[3] and an on-line voltage scheduling algorithm $\mathcal{A}$, we

[3]For the rest of this paper, we use a time-allocation tuple $\mathbf{A}$ (or $\mathcal{S}_{\mathbf{A}}$) instead of $\mathcal{S}$ (a function from time to the processor speed) to denote an off-line voltage schedule, because an off-line schedule $\mathbf{A}$ is a more appropriate representation for an off-line voltage schedule from the viewpoint of an on-line voltage scheduler.

use $\mathcal{A}\langle \mathbf{A}, (x_1, x_2, \cdots, x_{|\mathcal{J}|})\rangle$ to represent the on-line voltage schedule obtained by $\mathcal{A}$ when the actual workloads of $J_1, J_2, \cdots, J_{|\mathcal{J}|}$ are $x_1, x_2, \cdots, x_{|\mathcal{J}|}$ $(x_i \in (0, c_i])$, respectively. Given a feasible off-line schedule $\mathbf{A}$ that satisfies Condition III, an on-line voltage scheduling algorithm $\mathcal{A}$ is required to give each job the actual workload between its release time and deadline. Conversely, since $\mathcal{A}\langle \mathbf{A}, (c_1, c_2, \cdots, c_{|\mathcal{J}|})\rangle \equiv \mathcal{S}_{\mathbf{A}}$, an off-line schedule must satisfy Condition III. Therefore, the solution space for the WAOS problem (stated in terms of time-allocation tuples) is the same as that for the CWOS problem, and the WAOS is formulated as follows:

> Given workload probability distributions $\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_{|\mathcal{J}|}$ and an on-line scheduling algorithm $\mathcal{A}$, find an off-line schedule $\mathbf{A} = (a_1, a_2, \cdots, a_{|\mathcal{J}|})$ such that the average energy consumption
>
> $$\int_0^{c_{J_1}} \cdots \int_0^{c_{J_{|\mathcal{J}|}}} \left(\prod_{i=1}^{|\mathcal{J}|} \mathcal{P}_i(x_i) \right) \cdot E(\mathcal{A}\langle \mathbf{A}, (x_1, \cdots, x_{|\mathcal{J}|})\rangle)\ dx_{|\mathcal{J}|} \cdots dx_1 \tag{16}$$
>
> is minimized subject to Condition III (or Condition IV for an EDF job set).

Note that the CWOS problem is the special case of the WAOS problem where the workload of J_i is always c_i for all $1 \le i \le |\mathcal{J}|$.

The difficult of the CWOS comes mainly from how to express the integral (16) in terms of an off-line schedule $\mathbf{A} = (a_1, a_2, \cdots, a_{|\mathcal{J}|})$, because it requires an analysis on the runtime behavior of an arbitrary on-line scheduling algorithm. For this, we propose a generic on-line scheduling algorithm in Section 6.1. Then, we give an average-case analysis of the generic on-line algorithm, which gives an analytic expression of the integral (16) in terms of an off-line schedule. The analytic expression can be minimized by a modified version of the algorithm for the CWOS problem in conjunction with standard convex optimization technique [2].

6.1 A Generic On-Line Algorithm

In this section, we characterize on-line fixed-priority scheduling based on the behavior of the existing on-line voltage scheduling algorithms for fixed-priority job sets [17, 10, 4]. From the characterization, we construct a generic on-line voltage scheduling algorithm, which will be used as the base on-line scheduling algorithm for our workload-aware off-line scheduling algorithm in Section 6.2.

On-line algorithms use the "run-calculate-assign-run" strategy to determine the on-line voltage schedule: (1) run a current job, (2) when the job is completed, reclaim the unused processor time, called the *slack*, (3) pass a part of the accumulated slack on to the next job, (4) calculate the speed of the next job, and (5) run the next job. Existing on-line algorithms differ only in step (3), and can be characterized by how much slack is given to a job at its release time or resumption.

A generic on-line voltage scheduling algorithm is given in Figure 14. An on-line algorithm can be directly obtained by implementing procedure ALLOCATE_SLACK appropriately. For example, Gruian's algorithm can be obtained by adopting the ASAP policy in allocating the slack, i.e., giving all the accumulated slack time to the next activated job. On the other hand, Shin's algorithm [17] and Pillai's algorithm [10] are based on the ALAP policy.

Figures 15 and 16 illustrate the off-line and on-line scheduling, respectively. Initially, a job $J_i \in \mathcal{J}$ is given an allowed execution time a_i which is determined by the off-line scheduler. Figures 15.(a) and 15.(b) show an example job set and its off-line schedule, respectively. (For simplicity, we select an off-line schedule with a constant processor speed.) In Figure 15.(b) J_4 and J_5 are preempted by higher priority jobs, and their execution intervals are split into sub-intervals. For example, the execution interval of J_5 are split into sub-intervals with lengths $a_{5,1}$ and $a_{5,2}$ where $a_{5,1} + a_{5,2} = a_5$. Figure 15.(c) shows the actual workload of each job and slack times, which are not known off-line.

On-line scheduling algorithms reclaim the slack times and distribute them into jobs when the jobs are released or resumed as shown in Figure 16. Figure 16.(a) shows the on-line schedule obtained by Shin's algorithm [17] or Pillai's algorithm [10]. Note that no slack time is given to J_6 at its release time and J_4 at its resumption. The on-line schedule in Figure 16.(b) obtained by Gruian's algorithm [4] is flatter than the schedule in Figure 16.(a). Under the ASAP policy in allocating the slack, a job starts with a lower speed than under the ALAP policy and, consequently, the on-line schedule tends to be flatter resulting in the lower energy consumption.

```
procedure INITIALIZE
    (a_1, a_2, ..., a_|J|) := a feasible off-line schedule;
    slack := 0;  /* accumulated slack */
    for (i := 1 to J)
        workload_left_i := c_i;
        time_used_i := 0;  /* the total amount of CPU time used by J_i */
        slack_used_i := 0;  /* the total amount of slack given to J_i */
    end for
end procedure

procedure DURING_IDLE_TIME
    decrement slack by the idle time.
end procedure

procedure UPON_RELEASED_OR_RESUMED (J_i)
    slack_allocated := [ ALLOCATE_SLACK (J_i) ];
    decrement slack by slack_allocated.
    allowed_time_left := a_i + slack_used_i - time_used_i;
    increment slack_used_i by slack_allocated.
    run J_i at the speed workload_left_i / (slack_allocated + allowed_time_left).
end procedure

procedure DURING_EXECUTION (J_i)
    decrement workload_left_i.
    increment time_used_i.
end procedure

procedure UPON_COMPLETE (J_i)
    /* reclaim the slack time left by J_i */
    increment slack by a_i + slack_used_i - time_used_i.
end procedure
```

Figure 14. The generic on-line voltage scheduling algorithm. On-line algorithms are characterized by procedure ALLOCATE_SLACK, which allocates a part of the accumulated slack to a job released or resumed.

6.2 Workload-Aware Off-Line Scheduling

Based on the average-case analysis, we reduce the WAOS problem to the CWOS problem so that the FPTAS for the CWOS problem can be used for the WAOS problem with slight modification. Recall that it suffices to consider an algorithm that minimizes the integral (16) for an EDF job set; then, the FPTAS for the WAOS problem (with fixed-priority job sets as the problem instances) can be directly obtained.

First, we derive the average energy consumption in the integral (16) as a function of an off-line schedule $\mathbf{A} = (a_1, a_2, \cdots, a_{|\mathcal{J}|})$, based on the behavior of the generic on-line scheduling algorithm described in Section 6. Figure 17 shows an on-line schedule of a job J_i. Initially, J_i is given a_i as its allowed execution time by an off-line scheduler. J_i may be preempted by higher priority jobs, and its execution interval is split into n_i sub-intervals $\mathbf{I}_{i,1}, \mathbf{I}_{i,2}, \cdots, \mathbf{I}_{i,n_i}$ with lengths $a_{i,1}, a_{i,2}, \cdots, a_{i,n_i}$, respectively, where $a_i = \sum_{j=1}^{n_i} a_{i,j}$. Furthermore, $u_{i,j}$ is used to denote the slack time given to J_i at its j-th activation. When J_i starts its execution, the available time for J_i is $a_i + u_{i,1}$ and the speed $s_{i,1}$ for $\mathbf{I}_{i,1}$ is set to $c_i/(a_i + u_{i,1})$. Consequently, the amount of workload executed within $\mathbf{I}_{i,1}$ is given by

$$w_{i,1} = s_{i,1} \cdot (u_{i,1} + a_{i,1}) = c_i \cdot \frac{u_{i,1} + a_{i,1}}{u_{i,1} + a_i} \ .$$

Similarly, we can derive a recurrence for the speed $s_{i,j}$ in $\mathbf{I}_{i,j}$ and the amount of workload $w_{i,j}$ executed in $\mathbf{I}_{i,j}$. At J_i's j-th invocation, the available execution time is $u_{i,j} + a_i - \sum_{k=1}^{j-1} a_{i,k}$ and the amount of workload executed is $\sum_{k=1}^{j-1} w_{i,k}$. Therefore, we have

$$s_{i,j} = \frac{c_i - \sum_{k=1}^{j-1} w_{i,k}}{u_{i,j} + a_i - \sum_{k=1}^{j-1} a_{i,k}} \ , \ \ w_{i,j} = s_{i,j} \cdot (u_{i,j} + a_{i,j}) \ .$$

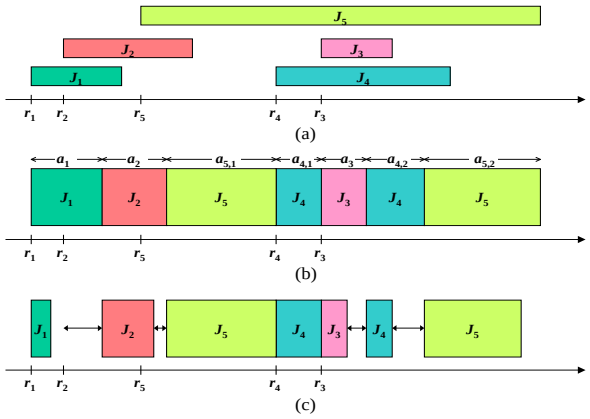


Figure 15. Off-line voltage scheduling examples; (a) an example job set, (b) an off-line schedule, and (c) the actual workload of each job. ($\leftrightarrow$ represents slack time.)

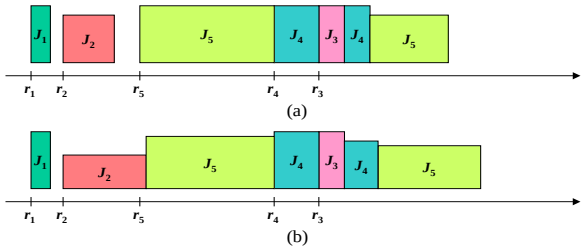


Figure 16. On-line voltage schedules obtained by (a) Shin's algorithm [18] or Pillai's algorithm [13], (b) Gruian's algorithm [5].

Solving the recurrence gives

$$s_{i,j} = \frac{c_i}{u_{i,1}+a_i} \cdot \prod_{l=1}^{j-1} \frac{a_i - \sum_{k=1}^{l} a_{i,k}}{u_{i,l+1} + a_i - \sum_{k=1}^{l} a_{i,k}} \quad , \quad w_{i,j} = c_i \cdot \frac{u_{i,j}+a_{i,j}}{u_{i,1}+a_i} \cdot \prod_{l=1}^{j-1} \frac{a_i - \sum_{k=1}^{l} a_{i,k}}{u_{i,l+1} + a_i - \sum_{k=1}^{l} a_{i,k}} \quad .$$

Let x_i be the actual workload of J_i ($x_i \leq c_i$) such that

$$\sum_{k=1}^{j-1} w_{i,k} \leq x_i < \sum_{k=1}^{j} w_{i,k} \quad .$$

Then, the total CPU time t_i used by J_i is given by

$$t_i \;=\; \sum_{k=1}^{j-1} (u_{i,k} + a_{i,k}) + \frac{x_i - \sum_{k=1}^{j-1} w_{i,k}}{s_{i,j}} \quad ,$$

and the energy consumed by J_i is given by

$$E_i(x_1, x_2, \cdots, x_{|\mathcal{J}|}) \;=\; \sum_{k=1}^{j-1} P(s_{i,k}) \cdot (u_{i,k} + a_{i,k}) \;-\; P(s_{i,j}) \cdot (t_i - \sum_{k=1}^{j-1} (u_{i,k} + a_{i,k})) \quad .$$

Now, it remains to derive $u_{i,j}$ in terms of an off-line schedule $(a_1, a_2, \cdots, a_{|\mathcal{J}|})$ and the actual workload of each job. In addition to these parameters, $u_{i,j}$ is dependent on the behavior of the base on-line scheduling algorithm, i.e., on how the slack is distributed by procedure ALLOCATE_SLACK in Figure 14. In this paper, we use the ASAP policy explained in Section 6, which showed the best energy efficiency in our experiment. Let J_{i_j} denote the job that completes its execution at J_i's j-invocation. Then, $u_{i,j}$ is given by

$$u_{i,j} = \sum_{k=1}^{n_{i_j}} (u_{i_j,k} + a_{i_j,k}) - t_{i_j} \quad .$$

We can now rewrite the average energy consumption E_{ave} (i.e., the integral (16)) as follows:

$$\sum_{i=1}^{|\mathcal{J}|} \int_0^{c_{J_1}} \cdots \int_0^{c_{J_{|\mathcal{J}|}}} \left(\prod_{i=1}^{|\mathcal{J}|} \mathcal{P}_{J_i}(x_i) \right) \cdot E_i(x_1, x_2, \cdots, x_{|\mathcal{J}|}) \, dx_{|\mathcal{J}|} \cdots dx_1$$

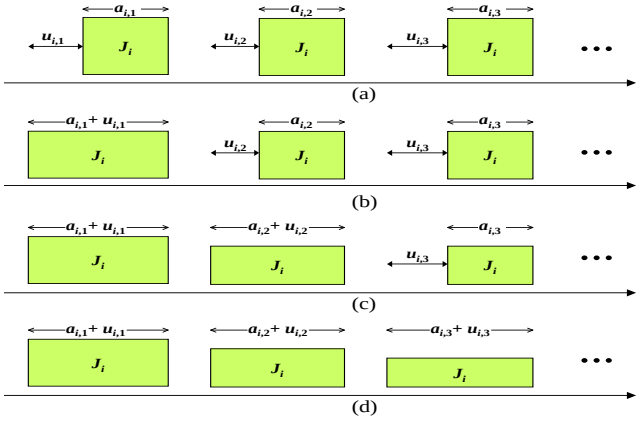


Figure 17. On-Line scheduling examples.

Note that the average energy consumption given in (17) can be proved to be a convex function of $(a_1, a_2, \cdots, a_{|\mathcal{J}|})$. Furthermore, since $(a_1, a_2, \cdots, a_{|\mathcal{J}|})$ is a convex set in $\mathbf{R}^{|\mathcal{J}|}$, the global minimum can be found by standard *convex optimization* technique [2]. As explained in Section 5, the FPTAS for the WAOS problem is directly obtained by substituting the convex optimization algorithm to minimize the average energy consumption for Yao's EDF voltage scheduling algorithm in the FPTAS for the CWOS problem.

7 Experimental Results

We evaluated the proposed algorithms for the CWOS problem and the WAOS problem. For both settings, we implemented the FPTAS in Section 5. We also implemented Yao's algorithm [22] and the convex optimization algorithm [2] for the CWOS problem and the WAOS problem, respectively. For the base on-line scheduling algorithm for the WAOS problem, we used the ASAP policy.

7.1 Evaluation of FPTAS for CWOS Problem

We first evaluated the FPTAS for the CWOS problem. For a comparison, we also implemented Quan's heuristic [12], which is currently the best polynomial-time voltage scheduling algorithm for fixed-priority real-time tasks. We compared the energy efficiency and computation time between two algorithms.[4] In our experiments, we assumed that the energy consumption is quadratically dependent on the supply voltage. For a given supply voltage V, the corresponding clock frequency f is proportional to $(V_{\mathrm{DD}} - V_{\mathrm{TH}})^{\alpha}/V_{\mathrm{DD}}$, where V_{TH} and α are assumed to be 0.5V and 1.3 [15].

We constructed test job sets from periodic task sets of three real-world applications: MPEG4 Videophone [16], CNC [6] and Avionics [9]. Table 1 summarizes the experimental results for these job sets. In each experiment, the execution time of each job (i.e., task instance) was randomly drawn from a Gaussian distribution[5] within the range of [WCET/10,WCET] of each task. Results were normalized over the energy consumption of each application scheduled by the proposed FPTAS with $\varepsilon = 0.1\%$. As shown in Table 1, the FPTAS outperforms Quan's algorithm spending reasonable CPU times. In the experiments, actual errors were always less than given ε's. (We omit CPU times for MPEG4 Videophone because they are less than 0.1 seconds.)

We also performed experiments using synthesized job sets with the varying number of jobs from 50 to 1600. We conjectured that one of the key parameters affecting the performance of Quan's algorithm is the degree of interferences among jobs. Since the degree of interferences is mainly dependent on the lengths of the execution intervals of the jobs, we generated three classes of job sets as follows: For the first class of job sets (**Class 1**), the release time and the length of the execution interval of a job are selected under the uniform distribution within [0,1000] and [50,100], respectively. The workload of each job was randomly selected from a uniform distribution within $[0.2, 1.0]$. (Note that it is sufficient to consider only the relative values of workloads since the maximum processor speed can be always appropriately adjusted.) For the second class of jobs (**Class 2**) and the third class

[4] We have implemented the exhaustive optimal algorithm by Quan *et al*. [13] as well for experiments. This algorithm, however, takes an excessive amount of time. For example, it took more than a day when $N = 25$. Therefore, we cannot include the experimental results for this algorithm.

[5] With the mean $m = \frac{\mathrm{WCET}/10 + \mathrm{WCET}}{2}$ and the standard deviation $\sigma = \frac{\mathrm{WCET} - \mathrm{WCET}/10}{6}$.

		Normalized Energy			CPU Time(s)	
Applications		MPEG4	CNC	Avionics	CNC	Avionics
# jobs		22	289	1372	289	1372
FPTAS	ε = 0.1%	1	1	1	44.71	4506.63
	ε = 0.5%	1.003	1.004	1.003	11.67	1021.48
	ε = 1.0%	1.006	1.008	1.007	6.12	631.15
	ε = 1.5%	1.012	1.013	1.011	5.16	512.32
	ε = 2.0%	1.017	1.018	1.018	3.81	313.15
Quan [12]		1.041	1.062	1.059	4.76	580.32

Table 1. Experimental results for three real-world real-time applications.

of jobs (**Class 3**), we used [100,300] and [300,500] (instead of [50,100]) for the length of the execution interval, respectively. Note that **Class 1**, **Class 2** and **Class 3** correspond to job sets with low, medium and high degrees of the interferences among the jobs. Tables 2, 3 and 4 show the experimental results for **Class 1**, **Class 2** and **Class 3**. As shown in tables, in general, the higher the degree of interferences becomes, the larger the improvement of our algorithm over Quan's algorithm becomes.

		Normalized Energy					
Job sets		$\mathcal{J}_1$	$\mathcal{J}_2$	$\mathcal{J}_3$	$\mathcal{J}_4$	$\mathcal{J}_5$	$\mathcal{J}_6$
# jobs		50	100	200	400	800	1600
FPTAS	ε = 0.1%	1	1	1	1	1	1
	ε = 0.5%	1.003	1.003	1.004	1.004	1.003	1.003
	ε = 1.0%	1.008	1.007	1.009	1.009	1.008	1.009
	ε = 1.5%	1.013	1.012	1.012	1.014	1.014	1.014
	ε = 2.0%	1.016	1.016	1.019	1.018	1.019	1.019
Quan [12]		1.044	1.047	1.051	1.054	1.052	1.071

Table 2. Experimental results for synthesized jobs (Class 1).

		Normalized Energy					
Job sets		$\mathcal{J}_1$	$\mathcal{J}_2$	$\mathcal{J}_3$	$\mathcal{J}_4$	$\mathcal{J}_5$	$\mathcal{J}_6$
# jobs		50	100	200	400	800	1600
FPTAS	ε = 0.1%	1	1	1	1	1	1
	ε = 0.5%	1.004	1.004	1.003	1.004	1.003	1.004
	ε = 1.0%	1.009	1.007	1.007	1.008	1.009	1.009
	ε = 1.5%	1.013	1.012	1.014	1.014	1.013	1.014
	ε = 2.0%	1.018	1.016	1.018	1.018	1.019	1.019
Quan [12]		1.055	1.062	1.070	1.079	1.103	1.127

Table 3. Experimental results for synthesized jobs (Class 2).

7.2 Evaluation of FPTAS for WAOS problem

We next evaluated the performance of the workload-aware off-line scheduling algorithm, i.e., the FPTAS for the WAOS problem. For a comparison, we experimented with three strategy. For the first strategy (**Strategy 1**), we used the workload-aware off-line scheduling algorithm and the base on-line scheduling algorithm as described in Section 6.2. The second strategy (**Strategy 2**) is identical to **Strategy 1** except that the workload-aware off-line algorithm is replaced by the FPTAS for the CWOS problem, which does not exploit the workload variation at off-line. For the third one (**Strategy 3**), we implemented the approach proposed by Pillai and Shin [10], which we believe is the best-known approach in the literature.

The real-world applications used in Section 7.1 was also used in the experiments and the results are collected in Table 5. In each experiment, the workload of each job (i.e., task instance) was randomly drawn from a Gaussian distribution[6] within the range of [BCET,WCET] of each job where BCET is the best case execution time. We performed the experiments by varying BCET from 10% to 90% of WCET for each application. Results were normalized over the energy consumption of each application scheduled by **Strategy 1**. As shown in Table 5, our workload-aware algorithm outperforms other approaches by up to about 40%.

[6] With the mean $m = \frac{\text{BCET}+\text{WCET}}{2}$ and the standard deviation $\sigma = \frac{\text{WCET}-\text{BCET}}{6}$.

		Normalized Energy					
Job sets		$\mathcal{J}_1$	$\mathcal{J}_2$	$\mathcal{J}_3$	$\mathcal{J}_4$	$\mathcal{J}_5$	$\mathcal{J}_6$
# jobs		50	100	200	400	800	1600
FPTAS	ε = 0.1%	1	1	1	1	1	1
	ε = 0.5%	1.004	1.004	1.004	1.003	1.004	1.004
	ε = 1.0%	1.009	1.007	1.007	1.009	1.008	1.009
	ε = 1.5%	1.014	1.013	1.014	1.013	1.014	1.014
	ε = 2.0%	1.018	1.017	1.019	1.018	1.019	1.019
Quan [12]		1.094	1.114	1.121	1.134	1.142	1.137

Table 4. Experimental results for synthesized jobs (Class 3).

		Strategy 2			Strategy 3		
Applications		MPEG4	CNC	Avionics	MPEG	CNC	Avionics
BCET/WCET	0.1	1.221	1.321	1.352	1.207	1.420	1.361
	0.3	1.117	1.214	1.193	1.124	1.311	1.252
	0.5	1.053	1.126	1.113	1.085	1.189	1.231
	0.7	1.019	1.046	1.041	1.022	1.098	1.071
	0.9	1.000	1.004	1.003	1.000	1.007	1.012

Table 5. Experimental results for average-case optimal FPTAS.

8 Conclusions

We investigated the problem of energy-optimal voltage scheduling for fixed-priority real-time systems implemented on a variable voltage processor. First, we proved the NP-hardness of the problem. Our complexity analysis provided an important new insight into the problem. Knowing the NP-hardness of the problem, as the best practical solution, we described a fully polynomial time approximation scheme (FPTAS) for the problem. That is, for any $\varepsilon > 0$, the proposed approximation scheme computes a voltage schedule whose energy consumption is bounded by $(1+\varepsilon)$ times that of the optimal voltage schedule with the running time bounded by a polynomial function of the number of input jobs and $1/\varepsilon$.

Furthermore, we present a workload-aware off-line scheduling approach which takes the workload variation and the behavior of an on-line scheduling algorithm into account. As part of the analysis of on-line algorithms, we propose a generic on-line voltage scheduling algorithm, which has its own significance in that it provides a new framework for the research on the on-line voltage scheduling. Experimental results show that our algorithm outperforms other well-known algorithms in the literature by up to about 40%.

While the proposed approximation scheme is efficient for general fixed-priority job sets, the proposed scheme can be further extended in several directions. For example, we are interested in devising more efficient algorithms for more specialized job sets such as job sets from periodic task sets. In addition, we plan to modify the proposed approximation scheme to work under a more realistic processor model with a limited number of voltage levels and voltage transition overheads.

References

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

[2] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. SIAM, 2001.

[3] M. Garey and D. Johnson. *Computers and Intractability*. W.H. Freeman and Company, 1979.

[4] 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.

[5] 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.

[6] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, and H. Shin. Visual Assessment of a Real-Time System Design: A Case Study on a CNC Controller. In *Proc. of Real-Time Systems Symposium*, pages 300–310, 1996.

[7] 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.

[8] W.-S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[9] C. Locke, D. Vogel, and T. Mesler. Building a Predictable Avionics Platform in Ada: A Case Study. In *Proc. of Real-Time Systems Symposium*, 1991.

[10] 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.

[11] Transmeta Corporation. Crusoe Processor. *http://www.transmeta.com*. 2000.

[12] G. Quan and X. Hu. Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors. In *Proc. of Design Automatioin Conference*, pages 828–833, 2001.

[13] G. Quan and X. Hu. An Optimal Voltage Schedule for Real-Time Systems on a Variable Voltage Processor. In *Proc. of Design, Automation and Test in Europe*, 2002.

[14] S. Sahni. Algorithms for Scheduling Independent Tasks. *Journal of the ACM*, 23:116–127, 1976.

[15] 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.

[16] D. Shin, J. Kim, and S. Lee. Low-Energy Intra-Task Voltage Scheduling Using Static Timing Analysis. In *Proc. of the 38th Design Automation Conference*, 2001.

[17] 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.

[18] Y. Shin, K. Choi, and T. Sakurai. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. In *Proc. of International Conference on Computer-Aided Design*, pages 365–368, 2000.

[19] AMD Corporation. PowerNow! Technology. *http://www.amd.com*. 2000.

[20] Intel Corporation. Intel XScale Technology. *http://developer.intel.com/design/intelxscale*. 2001.

[21] G. J. Woeginger. When Does a Dynamic Programming Formulation Guarantee the Existence of an FPTAS? In *Proc. of ACM-SIAM Symposium on Discrete Algorithms*, pages 820–829, 1999.

[22] 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.