Dynamic Speed Scaling Minimizing Expected Energy Consumption for Real-Time Tasks

Bruno Gaujal, Alain Girault, Stéphan Plassart

Noname manuscript No.
(will be inserted by the editor)

Dynamic Speed Scaling Minimizing Expected Energy Consumption for Real-Time Tasks

Gaujal Bruno · Girault Alain · Plassart Stephan

Received: date / Accepted: date

Abstract This paper proposes a Discrete Time Markov Decision Process (MDP) approach to compute the optimal on-line speed scaling policy to minimize the energy consumption of a single processor executing a finite or infinite set of jobs with real-time constraints. We provide several qualitative properties of the optimal policy: monotonicity with respect to the jobs parameters, comparison with on-line deterministic algorithms. Numerical experiments in several scenarios show that our proposition performs well when compared with off-line optimal solutions and out-performs on-line solutions oblivious to statistical information on the jobs.

Keywords Optimization · Real-Time Systems · Markov Decision Process · Dynamic Voltage and Frequency Scaling

This work has been partially supported by the LabEx PERSYVAL-Lab.

B. Gaujal
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France.
Tel.: +33 4 57 42 14 99
E-mail: bruno.gaujal@inria.fr

A. Girault
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France.
Tel.: +33 4 76 61 53 51
E-mail: alain.girault@inria.fr

S. Plassart
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France.
Tel.: +33 4 57 42 16 63
E-mail: stephan.plassart@inria.fr

1 Introduction

Minimizing the energy consumption of embedded system is becoming more and more important. This is due to the fact that more functionalities and better performances are expected from such systems, together with a need to limit the energy consumption, mainly because batteries are becoming the standard power supplies.

The starting point of this work is the seminal paper of Yao et al. [25] followed by the paper of Bansal et al. [3], both of which solve the following problem.

Let $(r_i, c_i, d_i)_{i\in\mathbb{N}}$ be a set of jobs, where r_i is the release date (or *arrival time*) of job i, c_i is its size (or *WCET*, or *workload*) *i.e.*, the number of processor cycles needed to complete the job, and d_i is its relative *deadline*, *i.e.*, the amount of time given to the processor to execute job i. The problem is to choose the

speed $s(t)$ of the processor[1] as a function of the time t, such that the processor can execute all the jobs before their deadlines, and such that the total energy consumption J is minimized. In this problem, J is the *dynamic* energy consumed by the processor: $J = \int_0^T j(s(t))dt$, where T is the time horizon of the problem (in the finite horizon case) and $j(s)$ is the power consumption when the speed is s.

This problem has been solved in Yao et al. [25] when the power function j is a *convex* function of the speed, in the *off-line* case, *i.e.*, when *all* jobs are known in advance. Many variants have been proposed to this off-line solution, for different job and energy models (see *e.g.*, [1]). However, in practice the *exact characteristics* of all the upcoming jobs cannot be known in advance, so the off-line case is unrealistic.

Several solutions have been investigated by Bansal et al. in [3] in the *on-line* case, *i.e.*, when only the jobs released at or before time t can be used to select the speed $s(t)$. Bansal et al. prove that an on-line algorithm introduced in [25], called Optimal Available (OA), has a competitive ratio of α^α when the power dissipated by the processor working at speed s is of the form $j(s) = s^\alpha$. In CMOS circuits, the value of α is typically 3. In such a case, (OA) may spend 27 times more energy than an optimal schedule in the worst case. The principle of (OA) is to choose, at each time t, the *smallest* processor speed such that all jobs released at or before time t meet their deadlines, under the assumption that no more jobs will arrive after time t.

However, this assumption made by (OA) is questionable. Indeed, the speed selected by (OA) at time t will certainly need to be compensated (*i.e.*, increased) in the future due to jobs released after t. This leads to an energetic inefficiency when the j function is convex. In contrast, our intuition is that the best choice is to select a speed *above* the one used by (OA) to *anticipate* on those future job arrivals.

The goal of our paper is to give a precise solution to this intuition by using *statistical knowledge* of the job arrivals (which could be provided by the user) in order to select the processor speed that optimizes the *expected* energy consumption.

Other constructions also based on statistical knowledge have been reported in [13,17,6] with a simpler framework, namely for a *single* job whose execution time is uncertain but whose release time and deadline are given, or in [21] by using heuristic schemes. Furthermore, [18] solves also an on-line problem, but with a task set of a *fixed* size; jobs have known execution times and deadlines, and their arrival times have known bounds. Moreover the scheduling policy of [18] is limited to the non-preemptive case. In contrast, we address the case of a finite or infinite number of jobs with uncertain release times, but with a known execution time at release time. This is a constrained optimization problem that we are able to model as an unconstrained Markov Decision Process (MDP) by choosing a proper state space that also encodes the constraints of the problem. This is achieved at the expense of the size of the state space (see § 2.4). In particular, this implies that the optimal speed at each time can be computed using a *dynamic programming* algorithm and that the optimal speed at any time t will be a deterministic function of the current state at time t.

In the first part of this paper (§ 2), we present our job model and the problem addressed in the paper. We define the state space of our problem (§ 2.3) and analyze its complexity (§ 2.4). In a second part (§ 3), we construct a Markov Decision Process model of this problem. We propose an explicit *dynamic programming* algorithm to solve it when the number of jobs is finite (§ 3.1), and a *Value Iteration* algorithm [22] for the infinite case (§ 3.2). Finally, we compute numerically the optimal policy in the finite and infinite horizon cases, and compare its performance with off-line policies

[1] Different communities use the term "speed" or "frequency", which are equivalent for a processor. In this paper, we use the term "speed".

and "myopic" policies like (OA), oblivious to the arrival of future jobs (§ 4). Moreover we present several useful generalizations: how to account for the cost of processor speed changes, for the cost of task context switches, and for non-convex power functions (§ 5).

2 Presentation of the Problem

2.1 Job features, Processor Speed, and Power

We consider a real-time system with one unicore processor that executes a set $\mathcal{J}$ of real-time jobs, sporadic and independent. In the finite case $\mathcal{J} = \{J_i\}_{1 \leq i \leq N}$ where N is the number of jobs, and in the infinite case $\mathcal{J} = \{J_i\}_{1 \leq i}$. Each job J_i is defined by the triplet (τ_i, c_i, d_i), where τ_i is the *inter-arrival time* between J_i and J_{i-1}, with $\tau_1 = 0$ by convention, c_i is the WCET, and d_i is the *relative deadline*. From the τ_i values, we can reconstruct the release time r_i of each job J_i as:

$$\begin{cases} r_1 = 0 & \text{by convention} \\ r_i = \sum_{k=1}^{i} \tau_k & \forall i > 1 \end{cases} \quad (1)$$

Jobs in $\mathcal{J}$ are ordered by their release times r_i, and jobs with the same arrival time are ordered arbitrarily.

We assume that the triplets $(\tau_i, c_i, d_i)_{i \in \mathcal{J}}$ are random variables, defined on a common probability space, whose joint distribution is known (for example by using past observations of the system): $\mathbb{P}(\tau_i = t, c_i = c, d_i = d)$ is supposed to be known for all t, c, d.

We also assume that the relative deadlines of the jobs are bounded by a maximal value, denoted Δ:

$$\Delta = \max_{i \in \mathcal{J}} \Delta_i \quad (2)$$

where Δ_i is the maximal value in the support of the distribution of the relative deadline d_i of job J_i, which is assumed to be finite. The assumption that the deadlines are bounded is classical in real-time systems.

Finally, we assume that the distribution of inter-arrival times has a finite memory bounded by L: For all $i \in \mathcal{J}$ and all t, c, d,

$$\begin{aligned} &\forall G \geq L, \\ &\mathbb{P}(\tau_i = t, c_i = c, d_i = d | \tau_i \geq G) \\ &\qquad = \mathbb{P}(\tau_i = t, c_i = c, d_i = d | \tau_i \geq L). \end{aligned} \quad (3)$$

We further define ℓ_t as the time elapsed between the last job arrival and t.

Regarding the single processor, we assume it can run at any time t at a speed $s(t)$ belonging to a finite set of integer speeds $\mathcal{S}$:

$$\forall t, s(t) \in \mathcal{S} = \{0, s_1, \ldots, s_{k-1}, s_{\max}\}.$$

The processor speeds are usually given as fractional numbers, *e.g.*, $\{0, 1/4, 1/2, 3/4, 1\}$, 1 being the maximal speed by convention. Without loss of generality, we scale the speeds such that $s_1, \ldots, s_k, s_{\max}$ are all integer numbers. For instance, the set $\{0, 1/4, 1/2, 3/4, 1\}$ will be scaled to $\{0, 1, 2, 3, 4\}$. This same scaling factor is also applied to the WCETs, *e.g.*, a job of size 1 becomes a job of size 4.

We consider that the power dissipated by the processor working at speed $s(t)$ at time t is $j(s(t))$, so that the energy consumption of the processor from time 0 to time T is computed as $J = \int_0^T j(s(t))dt$. Usually, the power consumption j is a convex increasing function of the speed (see [25,3]). This classical case is based on models of the power dissipation of CMOS circuits. Finer models use star-shaped functions [12] to further take into account static leakage. In the present paper, the function j is *arbitrary*. However several structural properties of the optimal speed selection will only hold when the function j is convex. In the numerical experiments (§ 4), several choices of j are used, to take into account different models of power consumption.

For the sake of simplicity, at first we only consider the following simple case: context switching time is null, speed changes are instantaneous and the power consumption function j is convex. However, preemption times, time lags for speed changes, as well as non-convex energy costs can be taken into account with minimal adaptation to the current model. A detailed

description of all these generalizations is provided in § 5.

2.2 Problem Statement

The objective is to choose at each time t the speed $s(t) \in \mathcal{S}$ in order to minimize the total energy consumption over the time horizon T, while satisfying the real-time constraints of all the jobs. Furthermore, the choice must be made on-line, *i.e.*, it can only be based on past and current information. In other words, only the jobs released at or before time t are known, while only statistical information is available for all future jobs.

As explained previously, the *statistical information* about the jobs is the distribution of the features: $\mathbb{P}(\tau_i = t, c_i = c, d_i = d)$ is supposed to be known for all t, c, d.

Besides, the *history* at time t is the set $\mathcal{H}(t)$ of all the jobs arrived at or before t, along with all the speeds used at or before t:

$$\mathcal{H}(t) = \{(\tau_i, c_i, d_i) | r_i \leq t\} \cup \{s(u), u \leq t\}. \quad (4)$$

Notice that in this model, unlike in [13,17], the workload c_i and the deadline d_i are known at the release time of job i[2].

We now define the on-line energy minimization problem $(\mathcal{P})$ as:

> *Find on-line speeds $s(t)$ (*i.e., *$s(t)$ can only depend on the history $\mathcal{H}(t)$) and a scheduling policy $R(t)$ in order to minimize the expected energy consumption under the constraint that no job misses its deadline.*

Since all release times and job sizes are integer numbers, the information available to the processor only changes at integer point.

In the following, we will focus on the case where the decision times (instants when the processor can change its speed) are also integers. It has been shown in [11] that this can be done without any loss in optimality. Now, if we consider that the speed $s(t)$ can only change at integer points too, we can focus on integer times: $t \in \mathbb{N}$ in the following.

Let (s^*, R^*) be an optimal solution to problem $(\mathcal{P})$. Since the energy consumption does not depend on the schedule (preemption is assumed to be energy-free) and since the *Earliest Deadline First* (EDF) scheduling policy is optimal for schedulability (see [15]), then (s^*, EDF) is also an optimal solution to problem $(\mathcal{P})$. In the following, we will always assume with no loss of optimality that the processor uses EDF to schedule its jobs. This implies that the only useful information to compute the optimal speed at time t, out of the whole history $\mathcal{H}(t)$, is simply the *remaining work*.

Definition 1 The remaining work at time t is an increasing function $w_t(\cdot)$ defined as follows: $w_t(u)$ is the amount of work arrived before t that must be completed before time $t + u$.

Since all available speeds, job sizes and deadlines are integer numbers, the remaining work $w_t(u)$ is an integer valued *càdlàg*[3] staircase function.

The definition of w_t is essential for the rest of the paper. Let us illustrate it in Figure 1, which shows the set of jobs released just before $t = 4$, namely $J_1 = (0, 2, 4)$, $J_2 = (1, 1, 5)$, $J_3 = (1, 2, 6)$, $J_4 = (1, 2, 4)$, $J_5 = (1, 0, 0)$, as well as the speeds chosen by the processor up to time $t = 4$: $s_0 = 1$, $s_1 = 0$, $s_2 = 2$, $s_3 = 1$. Function $A(t)$ is the amount of work that has arrived before time t. Function $D(t)$ is the amount of work that must be executed before time t. This requires a detailed explanation: the first step of $D(t)$ is the deadline of J_1 at $t = 0+4 = 4$; the second step is for J_2 at $t = 1+5 = 6$; the third step is for J_4 at $t = 3+4 = 7$; the fourth step is for J_3 at $t = 2+6 = 8$. Hence the step for J_4 occurs *before* the step for J_3. This is because Figure 1

[2] When the actual workload can be smaller than WCET, our approach still applies by modifying the state evolution Eq. (6), to take into account early termination of jobs.

[3] càdlàg = continue à droite, limite à gauche = right continuous with left limits.

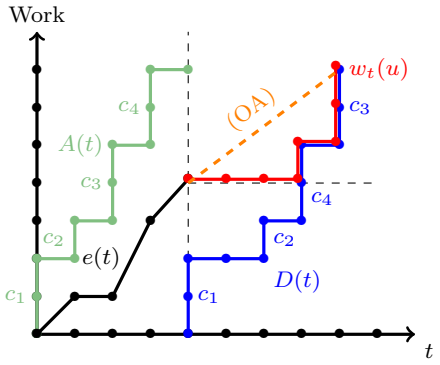

Fig. 1: Construction of the remaining work function $w_t(\cdot)$ at $t = 4$, for jobs $J_1 = (0, 2, 4)$, $J_2 = (1, 1, 5)$, $J_3 = (1, 2, 6)$, $J_4 = (1, 2, 4)$, $J_5 = (1, 0, 6)$, and processor speeds $s_0 = 1, s_1 = 0, s_2 = 2, s_3 = 1$. $A(t)$ is the amount of work that has arrived before time t. $D(t)$ is the amount of work that must be executed before time t. $e(t)$ is the amount of work already executed by the processor at time t.

depicts the situation at $t=4$. At $t=3$ we would only have seen the step for J_3. Finally, function $e(t)$ is the amount of work already executed by the processor at time t; in Figure 1, the depicted function $e(t)$ has been obtained with an arbitrary policy (*i.e.*, non optimal). Finally, the remaining work function $w_t(u)$ is exactly the portion of $D(t)$ that remains "above" $e(t)$. In Fig. 1, we have depicted in red the staircase function $w_t(u)$ for $t = 4$.

Remark 1 The on-line algorithm Optimal Available (OA) mentioned in the introduction is also based on the remaining work function: The speed of the processor at time t is the smallest slope of all linear functions above w_t. This is illustrated in Figure 1: the speed that (OA) would choose at time $t = 4$ is the slope of the green dotted line marked (OA); in the discrete speeds case (finite number of speeds), the chosen speed would be the smallest available speed just above the green dotted line.

Out of the whole history $\mathcal{H}(t)$, the remaining work function w_t together with the elapsed time since the latest arrival, ℓ_t, are the only relevant information at time t needed by the processor to choose its next speed. For this reason we call (w_t, ℓ_t) the *state* of the system at time t.

2.3 Description of the State Space

To formally describe the space $\mathcal{W}$ of all the possible remaining work functions and their evolution over time, we introduce several constructors.

Definition 2 We define the following operators:

- The time shift operator $\mathbb{T}f$ is the shift on the time axis of function f, defined as:
$$\forall t \in \mathbb{R}, \quad \mathbb{T}f(t) = f(t + 1).$$
- The positive part of a function f is $f^+ = \max(f, 0)$.
- The unit step function (Heaviside function), denoted $H_t(\cdot)$, is the discontinuous step function such that $\forall u \in \mathbb{R}$:
$$H_t(u) = \begin{cases} 0 & \text{if } u < t \\ 1 & \text{if } u \geq t. \end{cases}$$

Definition 3 The set $\mathcal{E}_t$ of the newly arrived jobs, released exactly at time t, is defined as:

$$\mathcal{E}_t = \{J_i = (\tau_i, c_i, d_i), i \in \mathbb{N} \,|\, r_i = t\}. \tag{5}$$

where r_i is the release time of job J_i, defined in Eq. (1).

Furthermore, to take into account the deadlines of the new jobs, we define in Def. 4 the function $a_t(u)$ that represents the work quantity arriving exactly at time t and that must be executed before time $t + u$. Formerly,

Definition 4 The new work arriving at t with absolute deadline before $t + u$ is given by the function $a_t(u) = \sum_{i \in \mathcal{E}_t} c_i H_{r_i+d_i}(t + u)$.

Lemma 1 *Let s_{t-1} be the processor speed at time $t - 1$. Then at time t the remaining work function becomes:*

$$w_t(\cdot) = \mathbb{T}\left[(w_{t-1}(\cdot) - s_{t-1})^+\right] + a_t(\cdot) \tag{6}$$

and the relationship between ℓ_t and ℓ_{t-1} is as follow:

$$\ell_t = \begin{cases} 0 & \text{if } \mathcal{E}_{t-1} \neq \emptyset \\ (\ell_{t-1} + 1) \wedge L & \text{otherwise.} \end{cases} \tag{7}$$

Proof Between $t-1$ and t, the processor working at speed s_{t-1} executes s_{t-1} amount of work, so the remaining work decreases by s_{t-1}. The remaining work cannot be negative by definition, hence the term $(w_{t-1}(\cdot) - s_{t-1})^+$. After a time shift by one unit, new jobs (belong to the set $\mathcal{E}_t$) are released at time t, bringing additional work, hence the additional term $a_t(\cdot)$.

Concerning ℓ_t, the time between the last job arrival and t, either there are some jobs that have arrived at $t-1$, *i.e.* $\mathcal{E}_{t-1} \neq \emptyset$, and in this case the last job arrival is at $t-1$, which implies $\ell_t = 1$, or no jobs have arrived at $t-1$, *i.e.* $\mathcal{E}_{t-1} = \emptyset$, and in this case the time delay increases, hence $\ell_t = \ell_{t-1}+1$ until ℓ_t reaches L, at which point, the exact value of ℓ_t becomes irrelevant. The only important information to assess the probability of future arrivals is the fact that ℓ_t is larger than L.

□

We illustrate in Fig. 2 the state change over an example, in the particular case where a single job arrives. The red line depicts the previous remaining work function $w_{r_{n-1}}$ at time r_{n-1}, while the blue line depicts the new remaining work function w_{r_n} following the arrival of the job $(1, c_n, d_n)$ at time r_n. The quantity of work executed by the processor is s_{n-1}.

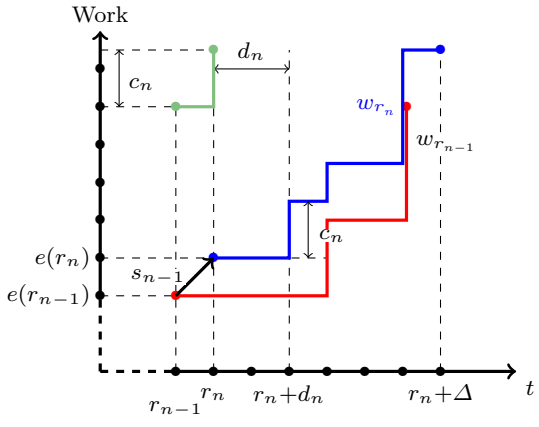

Fig. 2: State change following a job arrival at time r_n. The red line corresponds to the previous remaining work function. The blue line corresponds to the new remaining work function.

2.4 Size of $\mathcal{W}$

2.4.1 Feasible Jobs

Definition 5 (Feasibility) A set of jobs $\mathcal{S} = \{(\tau_i, c_i, d_i)\}_i$ is *feasible* if no deadline is missed when the processor always uses its maximal speed and the EDF schedule.

The processor can execute at most $ts_{\max}$ amount of work during a sliding time interval of size t. Since Δ is the maximal job deadline, all work arrived between t and $t+\Delta$ must be finished before $t+2\Delta$. The feasibility of jobs therefore requires that $2s_{\max}\Delta$ be an upper bound on the work quantity that can arrive between t and $t+\Delta$.

Let M be the maximal work quantity that can arrive during any sliding time interval of size Δ. According to the discussion above, the feasibility requirement implies that M must satisfy the following inequality:

$$M \leq 2s_{\max}\Delta \tag{8}$$

Therefore, feasibility implies that the size of the state space (equivalently, the number of remaining work functions) is finite. We compute precisely this state space in the next section.

2.4.2 Bound on the Size of $\mathcal{W}$

Proposition 1 *Let Δ be the maximal deadline of a job and $s_{\max}$ be the maximal speed. The size $Q(\Delta)$ of the set of remaining work functions $\mathcal{W}$ is bounded by:*

$$Q(\Delta) \leq \binom{\Delta s_{\max} + \Delta - 1}{\Delta - 1} \tag{9}$$

where the notation $\binom{n}{k}$ is the binomial coefficient.

Proof A state is an increasing integer functions $w_t(\cdot)$. As discussed before, in the worst case, the total remaining work at time t cannot exceed $\Delta s_{\max}$, and this remaining work is due before $t+\Delta$. Therefore, each remaining work

function can be seen, in the two-dimension integer grid, as an increasing path that connects the point $(0,0)$ to a point $(\Delta, K), K \leq \Delta s_{\max}$. Hence the size of the space $\mathcal{W}$ is smaller than the total number of increasing paths from $(0,0)$ to $(\Delta, \Delta s_{\max})$ (by extending paths ending in (Δ, K), with $K \leq \Delta s_{\max}$), that is:

$$Q(\Delta) \leq \binom{\Delta s_{\max} + \Delta - 1}{\Delta - 1}$$

□

2.4.3 Jobs with Bounded Sizes

Here we consider the particular case where the amount of work arriving at any time t is bounded (the bound is denoted by C). This leads to a smaller state space size, which is given in Prop. 2.

Proposition 2 *Let C be the maximal amount of work that can arrive at each time t. Then the size $Q(\Delta, C)$ of the space $\mathcal{W}$ is bounded by:*

$$Q(\Delta, C) = \sum_{y_1=0}^{C} \sum_{y_2=y_1}^{2C} \sum_{y_3=y_2}^{3C} \cdots \sum_{y_\Delta=y_{\Delta-1}}^{\Delta C} 1 \qquad (10)$$

It can be computed in closed form as:

$$Q(\Delta, C) = \frac{1}{1 + C(\Delta+1)} \binom{(C+1)(\Delta+1)}{\Delta+1} \qquad (11)$$

$$\approx \frac{e}{\sqrt{2\pi}} \frac{1}{(\Delta+1)^{3/2}} (e\,C)^{\Delta} \qquad (12)$$

Proof The proof is postponed to the Appendix.

The size of $\mathcal{W}$ will play a major role in the complexity of our dynamic programming algorithm to compute the optimal speeds.

3 Markov Decision Process (MDP) Solution

We denote by $\mathbf{x} = (w(\cdot), \ell)$ a *state* of the MDP, defined below. It is composed by a remaining work function denoted $w(\cdot)$, and the time elapsed since the latest job arrival denoted ℓ. We denote by $\mathcal{X}$ the state space (the set of all possible states).

As explained in § 2.4, the space $\mathcal{W}$ is finite and ℓ is bounded by L, so the set $\mathcal{X}$ is also finite. As a consequence, one can effectively compute the optimal speed in each possible state $\mathbf{x}$ using dynamic programming over $\mathcal{X}$.

In this section, we provide algorithms to compute the optimal speed selection in two cases: when the time horizon is finite and when it is infinite. In the finite case, we minimize the *total* energy consumption, while in infinite case we minimize the *long term average* energy consumed per time unit.

In both cases, we compute off-line the optimal *policy* σ_t^* that gives the speed the processor should use at time t in all its possible states. At runtime, the processor chooses at time t the speed $s(t)$ that corresponds to its current state $\mathbf{x}_t = (w_t, \ell_t)$, that is $s(t) := \sigma_t^*(\mathbf{x}_t)$.

The algorithms to compute the policy σ^* are based on a *Markovian evolution* of the jobs. From the distribution of jobs (τ_i, c_i, d_i), one can build, under state $\mathbf{x}$ and at time t, the distribution $(\phi_{\mathbf{x}})_{\mathbf{x}\in\mathcal{X}}$ of the work that arrives at t. For any $(c_1, \ldots, c_\Delta)$:

$$\phi_{\mathbf{x}}(t, c_1, \ldots, c_\Delta) = \mathbb{P}\left(a_t = \sum_{k=1}^{\Delta} c_k H_{k+t} \,|\, \mathbf{x}_t = \mathbf{x} \right)$$

Once ϕ is given, the transition matrix $P_t(\mathbf{x}, s, \mathbf{x}')$ from state $\mathbf{x} = (w, \ell)$ to $\mathbf{x}' = (w', \ell')$ when the speed chosen by the processor is s is:

$$P_t(\mathbf{x}, s, \mathbf{x}') = \begin{cases} \phi_{\mathbf{x}}(t, c_1, \ldots, c_\Delta) \\ \quad \text{if } w' = \mathbb{T}\left[(w-s)^+\right] + \sum_k c_k H_{k+t} \\ \quad \text{and } \ell' = \begin{cases} 0 & \text{if } (c_1 \cdots c_\Delta) \neq (0 \cdots 0) \\ (\ell+1) \wedge L & \text{otherwise} \end{cases} \\ 0 \qquad \text{otherwise} \end{cases}$$

This shows that the transition probability can be expressed as a function of the probability distributions of the jobs, through the distribution ϕ. If jobs are independent, then ϕ can be computed using a convolution of the job distributions.

3.1 Finite Case: Dynamic Programming

We suppose in this section that the time horizon is finite and equal to T. This implies that we only consider a finite number of jobs. The goal is to minimize the total expected energy consumption J^* over the time interval $[0, T]$. If the initial state is $\mathbf{x}_0$, then

$$J^*(\mathbf{x}_0) = \min_{\sigma} \left(\mathbb{E} \left(\sum_{t=0}^{T} j(\sigma_t(\mathbf{x}_t)) \right) \right) \tag{13}$$

where the expectation is taken over all possible job arriving sequences following the probability distribution of the features and where σ is taken over all possible *policies* of the processor: $\sigma_t(\mathbf{x})$ is the speed used at time t if the state is $\mathbf{x}$. The only constraint on $\sigma_t(\mathbf{x})$ is that it must belong to the set of available speeds, *i.e.*, $\sigma_t(\mathbf{x}) \in \mathcal{S}$, and it must be large enough to execute the remaining work at the next time step:

$$\forall t, \sigma_t(\mathbf{x}) \geq w(1) \tag{14}$$

The set of *admissible speeds* in state $\mathbf{x}$ is denoted $\mathcal{A}(\mathbf{x})$ and is therefore defined as:

$$\mathcal{A}(\mathbf{x}) = \left\{ s \in \mathcal{S} \text{ s.t. } s \geq w(1) \right\} \tag{15}$$

J^* can be computed using a backward induction. Let $J_t^*(\mathbf{x})$ be the minimal expected energy consumption from time t to T, if the state at time t is $\mathbf{x}$ ($\mathbf{x}_t = \mathbf{x}$). We present in the next section an algorithm to compute J^*.

3.1.1 Dynamic Programming Algorithm (DP)

We use a backward induction (Dynamic Programming) to recursively compute the expected energy consumption J^* and the optimal speed policy σ^*. We use the finite Horizon-Policy Evaluation Algorithm from [22] (p. 80). We obtain an *optimal policy* that gives the processor speed that one must apply in order to minimize the energy consumption (Algorithm 1).

The complexity to compute the optimal policy $\sigma_t^*(\mathbf{x})$ for all possible states and time steps is $\mathcal{O}(T|\mathcal{S}|Q(\Delta)^2)$. The combinatorial explosion of the state space makes it very large when the maximum deadline is large. Note however that this computation is done *off-line*. At runtime, the processor simply considers the current state $\mathbf{x}_t$ at time t and uses the pre-computed speed $\sigma_t^*(\mathbf{x}_t)$ to execute the job with the earliest deadline.

Algorithm 1 Dynamic Programming Algorithm (DP) to compute the optimal speed for each state and each time.

$t \leftarrow T$ % time horizon
for all $\mathbf{x} \in \mathcal{X}$ **do** $J_t^*(\mathbf{x}) \leftarrow 0$ **end for**
while $t \geq 1$ **do**
  for all $\mathbf{x} \in \mathcal{X}$ **do**
    $J_{t-1}^*(\mathbf{x}) \leftarrow \min_{s \in \mathcal{A}(\mathbf{x})} \Big(j(s) + \sum_{\mathbf{x}' \in \mathcal{W}} P_t(\mathbf{x}, s, \mathbf{x}') J_t^*(\mathbf{x}') \Big)$
    $\sigma_{t-1}^*[\mathbf{x}] \leftarrow \arg\min_{s \in \mathcal{A}(\mathbf{x})} \Big(j(s) + \sum_{\mathbf{x}' \in \mathcal{W}} P_t(\mathbf{x}, s, \mathbf{x}') J_t^*(\mathbf{x}') \Big)$
  end for
  $t \leftarrow t - 1$ % backward computation
end while
return all tables $\sigma_t^*[\cdot] \quad \forall t = 0 \dots T-1$.

3.1.2 Runtime Process: Table Look-UP (TLU_{DP})

At runtime, the processor computes the current state $\mathbf{x}_t$ and simply uses a Table Look-Up algorithm (TLU) to obtain its optimal speed $\sigma_t^*[\mathbf{x}_t]$, the speed tables having been computed off-line by (DP). This algorithm, called (TLU_{DP}), is shown in Algorithm 2. The size of the table is $T \times Q(\Delta)$ and the runtime cost for (TLU_{DP}) is $\mathcal{O}(1)$.

Algorithm 2 Runtime process (TLU_{DP}) used by the processor to apply the optimal speed

For Each $t = 0 \dots T-1$
  Update $\mathbf{x}_t$ using Eq. (6)
  Set $s := \sigma_t^*[\mathbf{x}_t]$
  Execute the job(s) with earliest deadline at speed s for one time unit
End

3.2 Infinite Case: Value Iteration

When the time horizon is infinite, the total energy consumption is also infinite whatever the speed policy. Instead of minimizing the total energy consumption, we minimize the *long term average* energy consumption per time-unit, denoted g. We therefore look for the optimal policy σ^* that minimizes g. In mathematical terms, we want to solve the following problem. Compute

$$g^* = \min_{\sigma} \mathbb{E}\left(\lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} j(\sigma(\mathbf{x}_t))\right) \tag{16}$$

under the constraint that no job misses its deadline.

3.2.1 Stationary Assumptions

In the following we will make the following additional assumption on the jobs: The size and the deadline of the next job have *stationary distributions* (*i.e.*, they do not depend on time). We further assume that the probability that no job arrives in the next time slot is strictly positive.

Under these two assumptions, the state space transition matrix is *unichain* (see [22] for a precise definition). Basically, the unichain property is true because, starting from an empty system (state $w_0 = (0, \ldots, 0)$), it is possible to go back to state w_0 no matter what speed choices have been made and what jobs have occurred. This is possible indeed because, with positive probability, no job will arrive for long enough a time so that all past deadlines have been met and the state goes back to w_0.

When the state space is unichain, the limit in Eq. (16) always exists (see [22]) and can be computed with an arbitrary precision using a *value iteration* algorithm (VI), presented in the next section.

3.2.2 Value Iteration Algorithm (VI)

The goal of Algorithm (VI) is to find a *stationary* policy σ (*i.e.*, σ will not depend on t), which is optimal, and to provide an approximation of the gain (*i.e.*, the average reward value g^*) with an arbitrary precision ε.

Algorithm 3 Value Iteration Algorithm (VI) to compute the optimal speeds in each state and the average energy cost per time unit.

$u^0 \leftarrow (0, 0, \ldots, 0)$, $u^1 \leftarrow (1, 0, \ldots, 0)$
$n \leftarrow 1$
$\varepsilon > 0$ % stopping criterion
while $span(u^n - u^{n-1}) \geq \varepsilon$ **do**
  for all $\mathbf{x} \in \mathcal{W}$ **do**
    $u^{n+1}(\mathbf{x}) \leftarrow \min_{s\in\mathcal{A}(\mathbf{x})} \Big(j(s) + \sum_{\mathbf{x}'\in\mathcal{W}} P(\mathbf{x}, s, \mathbf{x}')u^n(\mathbf{x}')\Big)$
  end for
  $n \leftarrow n + 1$
end while
Choose any $\mathbf{x} \in \mathcal{W}$ and let
$g^* \leftarrow u^n(\mathbf{x}) - u^{n-1}(\mathbf{x})$
for all $\mathbf{x} \in \mathcal{W}$ **do**
  $\sigma^*[\mathbf{x}] \leftarrow \arg\min_{s\in\mathcal{A}(\mathbf{x})} \Big(j(s) + \sum_{\mathbf{x}'\in\mathcal{W}} P(\mathbf{x}, s, \mathbf{x}')u^n(\mathbf{x}')\Big)$
end for
return σ^*

In Algorithm 3, the quantity u^n can be seen as the total energy up to iteration n. Moreover, the *span* of a vector z is the difference between its maximal value and its minimal value: $span(z) = \max_i(z_i) - \min_i(z_i)$. A vector with a span equal to 0 has all its coordinates equal.

Algorithm 3 computes both the optimal average energy consumption per time unit (g^*) with a precision ε as well as an ε-optimal speed to be selected in each state ($\sigma^*[\mathbf{x}]$).

The time complexity to compute the optimal policy depends exponentially on the precision $\frac{1}{\varepsilon}$. The numerical experiments show that convergence is reasonably fast (see § 4).

3.2.3 Runtime Process: Table Look-Up (TLU_{VI})

As for (TLU_{DP}), at each integer time $t \in \mathbb{N}$, the processor computes its current state $\mathbf{x}_t$ and retrieves its optimal speed $s := \sigma^*[\mathbf{x}]$ by

looking-up in the table σ^* that was pre-computed by (VI). This algorithm is identical to Algorithm 2, except for the the size of the table, which is $Q(\Delta)$.

3.3 Schedulability Issues

Let us recall that, according to Definition 5, a set of jobs is *feasible* if using the maximal speed $s_{\max}$ all the time, no job misses its deadline.

Notice that this is a condition on the jobs, unrelated to the speed policy of the processor.

Definition 6 (Schedulability) A set of jobs is *schedulable under speed policy* σ if, using speed $\sigma(\cdot)$, the processor executes all jobs without missing a deadline.

The goal of this section is to show the following result.

Proposition 3 *A finite (resp. infinite) set of jobs is feasible if and only if it is schedulable under policy* (DP) *(resp.* (VI)*).*

Proof We will first show that for all states $\mathbf{x}$ reached under (DP) (resp. (VI)) by executing a feasible set of jobs, the set of admissible speeds $\mathcal{A}(\mathbf{x})$ (defined in Eq. (15)) is never empty.

To show this, let us first modify the processor by allowing unbounded speeds, and let us introduce a new energy function $j(\cdot)$ such that $\forall s \geq s_{\max}$ we have $j(s) = \infty$. For speed values smaller than $s_{\max}$, the function $j(\cdot)$ remains unchanged. This modification is valid because one can assign an arbitrary energy consumption to unattainable speeds as one pleases: such speeds will never be used by valid speed policies. In this new framework, the processor can now use unbounded speeds:

$$\mathcal{S}' = \mathcal{S} \cup \{s_{\max} + 1, s_{\max} + 2, \ldots\}$$

but when it uses speeds higher than $s_{\max}$, its energy consumption becomes infinite.

Now, let us consider a set of feasible jobs executed by this extended processor model. Let us also define a simple policy, denoted $\sigma^{\max}$, that uses speed $s_{\max}$ at any time t:

$$\forall t, \sigma^{\max}(t) = s_{\max} \tag{17}$$

Under policy $\sigma^{\max}$, the expected (resp. long run average) energy consumption is $Tj(s_{\max})$ (resp. $j(s_{\max})$) and no job misses its deadline, by the definition of feasibility.

Since the optimal policy (DP) (resp. (VI)) is optimal in energy, it has a better expected (resp. long term) consumption than $\sigma^{\max}$. Hence, $J^*(\mathbf{x}_0)$ for (DP), as defined in § 3.1, (resp. g^* for (VI) in § 3.2) are finite. This implies that speeds higher than $s_{\max}$, that would have given an infinite energy consumption, were never used by the optimal policy.

To sum up, for any feasible set of jobs, the optimal policy never misses a deadline by construction of $\mathcal{A}(\mathbf{x}_t)$, and, according to the discussion above, it never uses a speed higher than $s_{\max}$. Therefore the optimal policy, as defined in Algorithm (1) for the finite case or Algorithm (3) for the infinite horizon case, will never miss a deadline if and only if the set of jobs is feasible. □

As a final remark, not all on-line policies will execute all jobs in a feasible set without missing deadlines when using speeds smaller than $s_{\max}$. For example, optimal available (OA), presented in § 3.5, requires additional constraints on a feasible set of jobs to guarantee schedulability (see [10]).

3.4 Bounded Job Sizes

As in § 2.4, let us assume that the amount of work that can arrive at any time t is bounded by C. In this case, one can assess more explicitly the feasibility condition of a set of jobs.

The necessary and sufficient feasibility condition of a set of jobs is:

$$s_{\max} \geq C \tag{18}$$

Indeed, if $s_{\max} < C$, then no speed policy can guarantee schedulability: a single job of size C

and relative deadline 1 cannot be executed before its deadline. The case where $s_{\max} = C$ is borderline because there exists a unique speed policy guaranteeing that no job will miss its deadline: at any time t, choose $s(t) = a_t(\Delta) \leq C$, where $a_t(\cdot)$ is the work quantity arrived at time t (see Def. 4).

If $s_{\max} > C$, then the previous policy never misses its deadline, hence using the discussion in the previous section, the optimal policy σ_t^* will also schedule all jobs before their deadline. This yields the following property.

Proposition 4 *Starting from an empty system, if the amount of work arriving at any time step is bounded by C, then schedulability with* (DP) *or* (VI) *is guaranteed if and only if $s_{\max} \geq C$.*

3.5 Properties of the Optimal Policy

In this section, we show several structural properties of the optimal policy σ^*, which are true for both the finite and infinite horizons.

3.5.1 Comparison with Optimal Available (OA)

Optimal Available (OA) is an on-line speed policy introduced in [25], which chooses the speed $s^{(\mathrm{OA})}(\mathbf{x}_t)$ at time t and in state $\mathbf{x}_t$ to be the minimal speed in order to execute the current remaining work at time t, should no further jobs arrive in the system. More precisely, at time t and in state $\mathbf{x}_t$, the (OA) policy uses the speed

$$s^{(\mathrm{OA})}(\mathbf{x}_t) = \max_u \frac{w_t(u)}{u} \tag{19}$$

where $w_t(\cdot)$ is the remaining work function computed by Eq. (6).

We first show that, under any state $\mathbf{x} \in \mathcal{X}$, the optimal speed $\sigma^*(\mathbf{x})$ is always higher than $s^{(\mathrm{OA})}(\mathbf{x})$.

Proposition 5 *Both in the finite or infinite case, the optimal speed policy σ^* satisfies*

$$\sigma^*(\mathbf{x}) \geq s^{(\mathrm{OA})}(\mathbf{x}) \tag{20}$$

for all state $\mathbf{x} \in \mathcal{X}$, if the power consumption j is a convex function of the speed.

Proof The proof is based on the observation that (OA) uses the optimal speed assuming that no new job will come in the future. Should some job arrive later, then the optimal speed will have to increase. We first prove the result when the set of speeds $\mathcal{S}$ is the whole *real* interval $[0, s_{\max}]$ (continuous speeds).

Two cases must be considered:

- If $s^{(\mathrm{OA})}(\mathbf{x}_t) = \max_u \frac{w_t(u)}{u}$ is reached for $u = 1$ (*i.e.*, $s^{(\mathrm{OA})}(\mathbf{x}_t) = w_t(1)$), then we have $\sigma^*(\mathbf{x}_t) \geq s^{(\mathrm{OA})}(\mathbf{x}_t)$ by definition, because the set of admissible speeds $\mathcal{A}(\mathbf{x}_t)$ only contains speeds higher than $w_t(1)$ (see Eq. (19)).
- If the maximum is reached for $u > 1$, then $\mathcal{A}(\mathbf{x}_t)$ may enable the use of speeds below $w_t(1)$.

Between time t and $t+u$, some new job may arrive. Therefore, the optimal policy should satisfy $\sum_{i=0}^{u-1} \sigma^*(\mathbf{x}_{t+i}) \geq w_t(u)$.

The convexity of the power function j implies[4] that the speeds in the optimal sequence $\sigma^*(\mathbf{x}_t), \ldots, \sigma^*(\mathbf{x}_{t+u-1})$ must all be above the average value $w_t(u)/u = s^{(\mathrm{OA})}(\mathbf{x}_t)$. In particular, for the first term, $\sigma^*(\mathbf{x}_t) \geq s^{(\mathrm{OA})}(\mathbf{x}_t)$.

Now, if the set of speeds is finite, then the optimal value of $\sigma^*(\mathbf{x}_t)$ must be one of the two available speeds in $\mathcal{S}$ surrounding $\sigma^{(\mathrm{OA})}(\mathbf{x}_t)$. Let s_1 and s_2 in $\mathcal{S}$ be these two speeds, *i.e.*, $s_1 < \sigma^{(\mathrm{OA})}(\mathbf{x}_t) \leq s_2$, and assume again that the max in Eq. (19) is not reached for $t = 1$. If the smallest speed s_1 is chosen as the optimal speed, this implies that further choices for $\sigma^*(\mathbf{x}_{t+i})$ will have to be greater or equal to s_2, to compensate for the work surplus resulting from choosing a speed below $\sigma^*(\mathbf{x}_t)$. This implies that it is never sub-optimal to choose s_2 in the first place (by convexity of the j function).

This trajectory based argument is true almost surely, so that the inequality $\sigma^*(\mathbf{x}_t) \geq$

[4] Actually, we use the fact that the sum $\sum_{i=0}^{u-1} j(s)$ is Schur-convex when j is convex (see [19]).

$s^{(\mathrm{OA})}(\mathbf{x}_t)$ will also hold for the *expected* energy over both a finite or infinite time horizon. □

3.5.2 Monotonicity Properties

Let us consider two sets of jobs $\mathcal{T}_1$ and $\mathcal{T}_2$ for which we want to apply our speed scaling procedure. We wonder which of the two sets uses more energy than the other when optimal speed scaling is used for both.

Of course, since jobs have random features, we cannot compare them directly, but instead we can compare their distributions. We assume in the following that the sizes and deadlines of the jobs in $\mathcal{T}_1$ (resp. $\mathcal{T}_2$) follow a distribution ϕ_1 (resp. ϕ_2) independent of the current state $\mathbf{x}$.

Definition 7 Let us define a stochastic order (denoted $\leq_s$) between the two sets of jobs $\mathcal{T}_1$ and $\mathcal{T}_2$ as follows. $\mathcal{T}_2 \leq_s \mathcal{T}_1$ if the respective distributions ϕ_1 and ϕ_2 are comparable. Formally, for any job (τ_1, c_1, d_1) with distribution ϕ_1 and any job (τ_2, c_2, d_2) with distribution ϕ_2, we must have:

$$\begin{aligned}&\forall\ \ \gamma, \delta,\ \ \mathbb{P}(c_2 \geq \gamma, d_2 \leq \delta) \leq \mathbb{P}(c_1 \geq \gamma, d_1 \leq \delta)\\ &\forall\ t \in \mathbb{N}, \mathbb{P}(\tau_1 = t) = \mathbb{P}(\tau_2 = t). \qquad (21)\end{aligned}$$

Moreover, by denoting $(i_1^1, \ldots, i_\Delta^1)$ the work quantity that arrives at time t for $\mathcal{P}_1$, and $(i_1^2, \ldots, i_\Delta^2)$ the work quantity that arrives at time t for $\mathcal{P}_2$, we have:

$$\begin{aligned}&\forall t, i_1^1, \ldots i_\Delta^1\ \ , i_1^2, \ldots i_\Delta^2,\\ &\quad \mathbb{P}(t, w_{t+1}\ \geq i_1^1, \ldots, w_{t+\Delta} \geq i_\Delta^\Delta)\\ &\qquad\qquad \leq \mathbb{P}(t, w_{t+1} \geq i_1^2, \ldots, w_{t+\Delta} \geq i_\Delta^2)\end{aligned}$$

Proposition 6 *If $\mathcal{T}_2 \leq_s \mathcal{T}_1$, then:*

1. *over a finite time horizon T, the total energy consumption satisfies $J^{(2)} \leq J^{(1)}$ (computed with Eq. (13));*
2. *in the infinite time horizon case, the average energy consumption per time unit satisfies $g^{(2)} \leq g^{(1)}$ (computed with Eq. (16)).*

Proof

Case 1 (finite horizon): The definition of $\mathcal{T}_2 \leq_s \mathcal{T}_1$ implies that we can couple the set of jobs $\mathcal{T}_1$ with the set of jobs $\mathcal{T}_2$, such that at each time $t \leq T$, $J_t^{(1)} = (\tau_t^1, c_t^1, d_t^1)$ and $J_t^{(2)} = (\tau_t^2, c_t^2, d_t^2)$ with $t = \tau_t^1 = \tau_t^2$, $c_t^2 \leq c_t^1$ and $d_t^2 \geq d_t^1$ (see [20]). It follows that the optimal sequence of speeds selected for $\mathcal{T}_1$ is admissible for $\mathcal{T}_2$, hence the optimal sequence for $\mathcal{T}_2$ should have a better performance. Since this is true for any set of jobs generated using ϕ_1, it is also true in expectation, hence $J^{(2)} \leq J^{(1)}$.

Case 2 (infinite horizon): We just use the fact that the optimal sequence for $\mathcal{T}_2$ is better than the optimal sequence for $\mathcal{T}_1$ over any finite horizon T. Letting T go to infinity shows that the average energy cost per time unit will also be better for $\mathcal{T}_2$. □

4 Numerical Experiments

4.1 Application Scenarios

Our approach is usable in several applicative contexts.

The first one concerns real-time systems whose tasks are *sporadic*, with no *a priori* structure on the job release times, sizes, and deadlines. In such a case, a long observation of the job features can be used to estimate the statistical properties of the jobs: distribution of the inter-release times, distribution of the job sizes, and deadlines.

Another case where our approach is efficient is for real-time systems consisting of several *periodic* tasks, each one with some randomly missing jobs. The uncertainty on the missing jobs may be due, for example, to faulty sensors and/or electromagnetic interference causing transmission losses in embedded systems.

A third situation is the case where jobs come from a high number of *periodic* tasks and each of them has an unknown jitter. If we suppose that we have a probabilistic knowledge of the jitter values, then we can use our model to improve the energy consumption by determining more quickly all the jitters of each task.

The last example is the case where jobs are produced by event-triggered sensors: This

case is also a superimposition of *sporadic* tasks, where the job probabilities represents the occurrence probability of events.

These examples are explored in this experimental section where our solution is compared with other on-line solutions. Our numerical simulations report a 5% improvement over (OA) in the sporadic tasks case, and 30% to 50% improvement in the periodic tasks case.

The numerical experiments are divided in two cases: In § 4.3.1 and § 4.3.2, we consider a real-time system with a single periodic task of period 1 with jobs that have randomness on their sizes[5].

The second set of experiments deals with another type of real-time systems made of several periodic tasks. Each task is characterized by its offset, period, size, and deadline. There is a randomness on the job size, that is due to sensor perturbation.

All the experiments reported below are based on these two scenarios.

4.2 Implementation Issues

The state space $\mathcal{X}$ has a rather complex structure and is very large. Therefore, the data structure used in the implementations of Algorithms 1 and 3 must be very efficient to traverse the state space as well as to address each particular state when state changes occur. This is done by using a hashing table to retrieve states according to a multi-dimensional *key* that represents the state, that is, the vector $[w(1), w(2)-w(1), \ldots, w(\Delta) - \sum_{k=1}^{\Delta-1} w(k)]$, and a recursive procedure based on Eq. (10) to traverse the state space.

The implementation of Algorithms 1 and 3 has been done in *R* to take advantage of the possibility to manipulate linear algebraic operation easily, and in *C* when the state space was too large to be efficiently handled in *R*.

[5] An estimation of the distribution of their size can be obtained through the measurement of many traces of the real-time system.

4.3 Experimental Set-up, Finite Case

Our experiments are done in two steps:

- Firstly, we compute the optimal speeds for each possible state $\mathbf{x} \in \mathcal{X}$. For this, we use Algorithm 1 (DP) or 3 (VI), and we store in the σ^* table the optimal speed for each possible state of the system.
- Secondly, to compare different speed policies, we simulate a sequence of jobs (produced by our real-time tasks, see § 4.1) over which we use our ($\mathrm{TLU}_{\mathrm{DP}}$) solution or other solutions (*e.g.*, off-line or (OA)) and we compute the corresponding energy consumption.

In a nutshell, the experiments show that our MDP solution performs very well in practice, almost as well as the optimal *off-line* solution (see § 4.3.1). Regarding the comparison with (OA), in most of our experiments, ($\mathrm{TLU}_{\mathrm{DP}}$) outperforms (OA) by 5% on average in the sporadic case when job inter-arrival times are i.i.d.[6] (see § 4.3.2). In the periodic case, where jobs are more predictable, the gap with (OA) grows to about 50% (see § 4.3.3).

4.3.1 Comparison with the Off-line Solution

To evaluate our on-line algorithm, we compare it with the off-line solution computed on a simulated set of jobs, characteristics of which are described in Example 1. We draw the aggregated work done by the processor (the respective speeds are the slopes) in two cases:

- The optimal off-line solution that only uses speeds in the finite set $\mathcal{S}$.
- The ($\mathrm{TLU}_{\mathrm{DP}}$) solution.

Example 1 One periodic task $\boldsymbol{T}_1$ of period 1 with jobs of variable size $c_1 = \{0, 2\}$ with respective probabilities (w.r.p.) $\{0.4, 0.6\}$ and deadline $d_1 = 5$. The processor can use 4 speeds $\mathcal{S} = \{0, 1, 2, 5\}$ and its energy consumption per time unit is given by the function $j(s) = s^3$.

[6] i.i.d. = independent and identically distributed random variables.

A job of size 0 at some time instant t is the same as no job at all at time t. In Example 1, the variable size $c_1 = \{0, 2\}$ actually models a *sporadic* task: with probability 0.4 no job arrives, while with probability 0.6 a job of size $c_1 = 2$ arrives.

In Example 1, the maximal speed is large enough so that schedulability is not an issue: $5 = s_{\max} > C = 2$ (§3.3). Note that, in contrast with (TLU_{DP}), some jobs created by task $\boldsymbol{T}_1$ might not be schedulable under (OA).

The result over one typical simulation of run for Example 1 is displayed in Figure 3. As expected, (TLU_{DP}) consumes more energy than the off-line case. The differences in the chosen speeds are the following: (i) speed 0 is used once by (TLU_{DP}) but is never used by the off-line solution; (ii) speed 2 is used 5 times by (TLU_{DP}) and only 4 times in the off-line case. The energy consumption gap between the two is $2^3 + 0^3 - 1^3 - 1^3 = 6$ J. The total energy consumption under the off-line solution is 46 J, while the total energy consumption under the (TLU_{DP}) solution is 52 J, the difference being 13% of the total energy consumption.

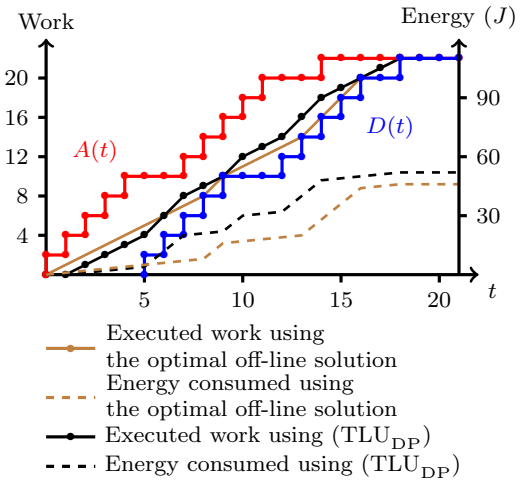

Fig. 3: Comparison of the executed work of off-line and (TLU_{DP}) solution on one simulation of Example 1. As defined before, $A(t)$, the red curve, is the workload arrived between 0 and T, and $D(t)$, the blue curve, is the workload deadlines from 0 to T; Brown curve: work executed using the optimal off-line speeds; Black curve: work executed using the speed selection computed by (TLU_{DP}).

4.3.2 Comparison with (OA)*, Sporadic Tasks*

Recall that, under (OA), the processor speed at time t in state $\mathbf{x}_t$ is set to $\max_u \frac{w_t(u)}{u}$. However, when the number of speeds is finite, the speed computed by (OA) might not be available. Hence, the speed $s^{(\text{OA})}(\mathbf{x}_t)$ chosen by (OA) must be set to the smallest speed in $\mathcal{S}$ greater than $\max_u \frac{w_t(u)}{u}$.

As a consequence, to compare (OA) and (DP), the number of possible speeds must be large enough to get a chance to see a difference between the two. We do so with Example 2, which consists of two sporadic tasks, using the same modeling technique as in Example 1 by fixing $c_2 = \{0, 3, 6\}$.

Example 2 One periodic task $\boldsymbol{T}_2$ of period 1, variable job size $c_2 = \{0, 3, 6\}$ w.r.p. $\{0.2, 0.6, 0.2\}$, and fixed deadline $d_2 = 3$. The processor can use 5 processor speeds $\mathcal{S} = \{0, 1, 2, 3, 4\}$ and its energy consumption per time unit follows the function $j(s) = s^3$.

We ran an exhaustive experiment consisting of 10,000 simulations of sequences of jobs generated by this periodic task, over which we computed the relative energy gain of (TLU_{DP}) over (OA) in percentage. The gain percentage of (TLU_{DP}) was in the range $[5.17, 5.39]$ with a 95% confidence interval and an average value of 5.28%.

Even if this gain is not very high, one should keep in mind that it comes for free once the (DP) solution has been computed. Indeed, using (TLU_{DP}) on-line takes a constant time to select the speed (table look-up) while using (OA) on-line takes $\mathcal{O}(\Delta)$ to compute the value $\max_u \frac{w_t(u)}{u}$.

Figure 4 shows a comparison between (OA) and (TLU_{DP}). The total work executed by the (TLU_{DP}) solution is always above the total work executed by (OA), as stated in Proposition 5. Moreover, the consumed energy is more important at the beginning with (TLU_{DP}) than

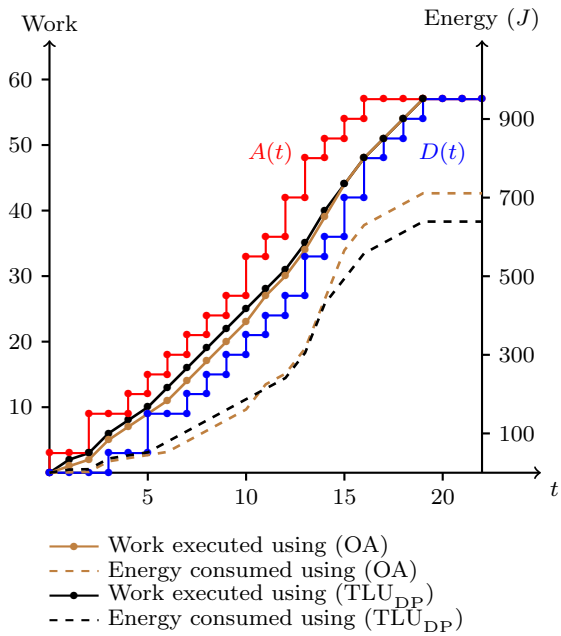

Fig. 4: Comparison of the executed work between (OA) and (TLU_{DP}) solutions, with fixed deadlines $d_n = 3$, size $c_n = \{0, 2, 4\}$ w.r.p. $\{0.2, 0.6, 0.2\}$, and processor speeds in $\{0, 1, 2, 3, 4\}$. As defined before, $A(t)$ (red curve) is the workload arrived between 0 and T, while $D(t)$ (blue curve) is the workload deadlines from 0 to T.

with (OA), because we anticipate the work that will arrive in the future. The processor executes more work so it consumes more energy with (TLU_{DP}) before time $t = 11$; but after this time, it's the opposite, the energy consumed by (TLU_{DP}) is lower than the energy consumed by (OA). Over the whole period, (TLU_{DP}) outperforms (OA): The total energy consumption for (OA) is 711 J (dashed brown curve) while that for (TLU_{DP}) is 639 J (dashed black curve). As a result, (TLU_{DP}) outperforms (OA) by a margin of around 10%. Even if this gain is not very high, one should keep in mind that, again, it comes for free once the (DP) solution has been computed off-line.

4.3.3 Comparison with (OA)*, Periodic Tasks*

We now consider several examples consisting of two or more periodic tasks. The fact that the probability matrix, which represents the state change, depends on the time is important in this section. Indeed, at each time step, the probability of the job arrival depends on the time and in particular on the modulo of the number of the considered task. For instance in Example 3 (see below), we have a probability that depends of the time instant modulo 2: at even time steps ($t = 0 \mod 2$), we have some probability p_1 that the job J_1 produced by task $\boldsymbol{T}_1$ arrives and the job J_2 produced by task $\boldsymbol{T}_2$ arrives with a probability equal to zero. In contrast, at odd time steps ($t = 1 \mod 2$), we have some probability p_2 that the job J_2 arrives and the job J_1 arrive with a probability equal to zero. On the other examples (Examples 4 and 5), we can perform the same analysis as above to show that the probability matrix depends on the time.

This Section displays three examples, Examples 3, 4, and 5, which consider different cases where we have several periodic tasks that have not necessarily the same offset and the same periodicity.

Example 3 Two periodic tasks $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$. For task $\boldsymbol{T}_1$, the period is 2, the offset is 0, with jobs of variable size $c_1 = \{0, 2\}$ w.r.p. $\{0.2, 0.8\}$, and the deadline is $d_1 = 2$. For task $\boldsymbol{T}_2$, the period is 2, the offset is 0, with jobs of variable size $c_1 = \{0, 4\}$ w.r.p. $\{0.25, 0.75\}$, and the deadline is $d_1 = 1$.

The total energy consumption over the 20 units of time is of 513 J for (TLU_{DP}), and 825 J for (OA), so more than 60% bigger. Here (TLU_{DP}) has a clear advantage because the job characteristics are highly predictable.

Example 4 Four periodic tasks $\boldsymbol{T}_1, \boldsymbol{T}_2, \boldsymbol{T}_3, \boldsymbol{T}_4$ with the same period equal to 4 and respective offsets $0, 1, 2, 3$. For each task $\boldsymbol{T}_i$, the job size is variable and deadline is fixed, with $c_1 = \{0, 2\}$ w.r.p. $\{0.2, 0.8\}$, and $d_1 = 2$; $c_2 = \{0, 1\}$ w.r.p. $\{0.2, 0.8\}$, and $d_2 = 3$; $c_3 = \{0, 4\}$ w.r.p. $\{0.2, 0.8\}$, and $d_3 = 2$; $c_4 = \{0, 2\}$ w.r.p. $\{0.2, 0.8\}$, and $d_4 = 1$.

With Example 4, the energy consumed by (TLU_{DP}) is on average 30% lower than the energy consumed by (OA). We performed 10,000 simulations over 40 time steps: the average gain is 29.04% with the following confidence interval at 95%: $[28.84, 29.24]$.

Example 5 Seven periodic tasks $\boldsymbol{T}_1$ to $\boldsymbol{T}_7$. Task $\boldsymbol{T}_4$ has period 4, offset 3, and variable job size $c_4 = \{0, 4\}$ w.r.p. $\{0.2, 0.8\}$, and $d_4 = 2$. All the other tasks $\boldsymbol{T}_1, \ldots, \boldsymbol{T}_3$ and $\boldsymbol{T}_5, \ldots, \boldsymbol{T}_7$ have period 8, respective offsets $0, 1, 2, 4, 5, 6, 7$ (8 being for the second job of $\boldsymbol{T}_4$), and respective parameters $c_1 = \{0, 2\}$ w.r.p. $\{0.2, 0.8\}$, and $d_1 = 1$; $c_2 = \{0, 1\}$ w.r.p. $\{0.2, 0.8\}$, and $d_2 = 2$; $c_3 = \{0, 1\}$ w.r.p. $\{0.2, 0.8\}$, and $d_3 = 3$; $c_5 = \{0, 4\}$ w.r.p. $\{0.2, 0.8\}$, and $d_5 = 1$; $c_6 = \{0, 2\}$ w.r.p. $\{0.2, 0.8\}$, and $d_6 = 2$; $c_7 = \{0, 4\}$ w.r.p. $\{0.2, 0.8\}$, and $d_7 = 3$.

With Example 5, the energy consumed by (TLU_{DP}) is on average 47% lower than the energy consumed by (OA). We performed 10,000 simulations over 80 time steps, the average gain was 46.88% with the following confidence interval at 95%: $[46.71, 47.04]$.

The other simulation parameters for Examples 3 to 5 are $T = 20$, $\mathcal{S} = \{0, 1, 2, 3, 4, 5\}$ and $j(s) = s^3$.

Table 1 summarizes these results.

Table 1: Comparisons between (OA) and (TLU_{DP}).

example	gain over (OA)	95% confidence interval
Ex. 3 (2 tasks)	56.44%	[56.21, 56.68]
Ex. 4 (4 tasks)	29.04%	[28.84, 29.24]
Ex. 5 (7 tasks)	46.88%	[46.71, 47.04]

In all these cases, (TLU_{DP}) outperforms (OA) by a greater margin than with sporadic tasks. The reason is that the job sequence is more predictable, so the statistical knowledge over which (TLU_{DP}) is based is more useful here than in the sporadic case.

4.4 Computation Experiments, Infinite Case

In this section, we run algorithm (VI) (Algorithm 3) to compute the optimal speed to be used at each time step over an infinite horizon. We fix the stopping criterion in Algorithm 3 to $\varepsilon = 1.0 * 10^{-5}$, so our computation of the average energy consumption is precise by at least 5 digits. We ran the program in the following two cases:

Example 6 One periodic task of period 1 with jobs of variable size $c_6 = \{0, 2\}$ w.r.p. $\{1-p, p\}$, and fixed deadline $d_6 = 3$, with p varying from 0 to 1.

Example 7 One periodic task of period 1 with jobs of variable size $c_7 = \{0, 2\}$ w.r.p. $\{1-p, p\}$, and fixed deadline $d_7 = 5$, with p varying from 0 to 1.

In both examples, the available processor speeds are in the set $\mathcal{S} = \{0, 1, 2\}$ and the energy consumption function is $j(s) = s^2$. The only difference between Examples 6 and 7 are the deadlines.

The results of our computations are displayed in Figure 5. The three curves depict respectively the average energy consumption per time unit as a function of the probability p (which varies from 0 to 1) for Examples 6 and 7, together with the theoretical lower bound.

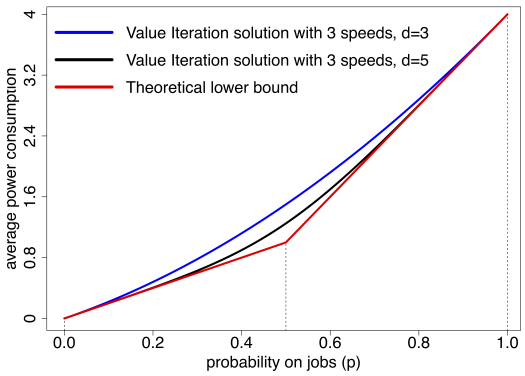

Fig. 5: Average energy consumption per time unit for (TLU_{VI}): theoretical lower bound (red curve), deadlines equal to 3 (black curve, Example 6), and deadline equal to 5 (blue curve, Example 7).

The different curves in Figure 5 have the following meaning:

- The black and blue curves correspond to the (VI) solution with three processor speeds $\mathcal{S} = \{0, 1, 2\}$. These curves display g^* (computed by Algorithm 3) as a function of p, the probability that a job of size $c = 2$ and deadline $d = 3$ (black curve) or deadline $d = 5$ (blue curve) arrives in the next instant.
- The red curve is the theoretical lower bound on g^*, oblivious of the jobs distribution and deadlines, only based on the average amount of work arriving at each time slot.

As expected according to Proposition 6, the higher the arrival rate, the higher the average energy consumption: both curves are increasing.

Proposition 6 also implies that larger deadlines improve the energy consumption. This is in accordance with the fact that the black curve (deadline 5) is below the blue curve (deadline 3).

What is more surprising here is how well our solution behaves when the deadline is 5. Its performance is almost indistinguishable from the theoretical lower bound (valid for all deadlines) over a large range of the rate p. More precisely, the gap between our solution with deadline equal to 5 and the theoretical lower bound is less than 10^{-3} for $p \in [0, 0.20] \cup [0.80, 1]$.

4.4.1 Lower Bound

The theoretical lower bound has been obtained by solving the optimization problem without taking into account the distribution of the jobs features nor the constraint on the deadlines. Without constraints, and since the power is a convex function of the speed (here $j(s) = s^2$), the best choice is to keep the speed constant. The ideal constant speed needed to execute the jobs over a finite interval $[0, T]$ is $A(T)/T$, where $A(T)$ is the workload arrived before T. When T goes to infinity, the quantity $A(T)/T$ converges to $2p$ by the strong law of large numbers. Therefore, the optimal constant speed is $s^\infty = 2p$.

Now, if we consider the fact that only 3 processor speeds, namely $\{0, 1, 2\}$, are available, then the ideal constant speed $s^\infty = 2p$ cannot be used. In this case, the computation of the lower bound is based on the following construction.

On the one hand, if $0 \leq p \leq \frac{1}{2}$, then the ideal constant processor speed, $s^\infty = 2p$, belongs to the interval $[0, 1]$. In that case, only speeds $\{0, 1\}$ will be used. To obtain an average speed equal to $2p$, the processor must use speed 1 during a fraction $2p$ of the time and the speed 0 the rest of the time. The corresponding average energy per time unit has therefore the following form:

$$g^\infty = 2p \times 1^2 + (1 - 2p) \times 0^2 = 2p \tag{22}$$

On the other hand, if $p \geq \frac{1}{2}$, then the ideal constant processor speed, $s^\infty = 2p$, belongs to the interval $[1, 2]$. In that case, the processor only uses speeds 1 or 2. To get an average speed of $2p$, the processor must use the speed 2 during a fraction $2p - 1$ of the time and the speed 1 the rest of the time. The corresponding average energy per time unit in this case is:

$$g^\infty = (2p-1) \times 2^2 + (2-2p) \times 1^2 = 6p-2 \tag{23}$$

By combining Eqs. (22) and (23), we obtain the lower bound on g:

$$g^\infty(p) = \begin{cases} 2p & \text{if } p \leq 1/2 \\ 6p - 2 & \text{if } p \geq 1/2 \end{cases} \tag{24}$$

This is the red curve in Figure 5.

4.4.2 Comparison of ($\mathrm{TLU}_{\mathrm{DP}}$) *and* ($\mathrm{TLU}_{\mathrm{VI}}$)

We performed a comparison between the two algorithms ($\mathrm{TLU}_{\mathrm{DP}}$) and ($\mathrm{TLU}_{\mathrm{VI}}$) over different time horizons T in order to study the impact of this parameter. The gain in energy of ($\mathrm{TLU}_{\mathrm{DP}}$) vs ($\mathrm{TLU}_{\mathrm{VI}}$), represented in blue in Figure 6, is computed as follow:

$$\frac{\mathtt{Energy}_{(\mathrm{VI})} - \mathtt{Energy}_{(\mathrm{DP})}}{\mathtt{Energy}_{(\mathrm{DP})}} \tag{25}$$

This fraction computes the relative difference between the infinite horizon case algorithm (Algorithm 3) and the the finite horizon case algorithm (Algorithm 1). Besides, the cost of (OA) versus ($\mathrm{TLU}_{\mathrm{DP}}$), also represented in Figure 6 as a dashed red curve, is defined as follows:

$$\frac{\texttt{Energy}_{(\mathrm{OA})} - \texttt{Energy}_{(\mathrm{DP})}}{\texttt{Energy}_{(\mathrm{DP})}} \tag{26}$$

Computations were done on Example 2 with 10,000 simulations. They are summarized in Table 2.

Table 2: Influence of the time horizon T on ($\mathrm{TLU}_{\mathrm{DP}}$) in comparison with ($\mathrm{TLU}_{\mathrm{VI}}$).

T	10	15	20	25
(VI) vs (DP)	4.3%	1.7%	0.95%	0.62%
(OA) vs (DP)	4.8%	5.3%	5.3%	5.2%
T	30	40	100	150
(VI) vs (DP)	0.45%	0.29%	0.099%	0.064%
(OA) vs (DP)	5.0%	4.9%	4.6%	4.5%
T	200	250	1000	
(VI) vs (DP)	0.046%	0.031%	$6.19{\cdot}10^{-5}$%	
(OA) vs (DP)	4.3%	4.3%	4.2%	

One can notice in Table 2 as well as on the blue curve in Figure 6 that, as soon as the time horizon is greater than 20 time units, the energy difference between ($\mathrm{TLU}_{\mathrm{DP}}$) and ($\mathrm{TLU}_{\mathrm{VI}}$) is smaller than 1%, and is negligible in comparison with the energy difference between ($\mathrm{TLU}_{\mathrm{DP}}$) and (OA). We conclude that using (VI) instead of (DP) is a good approximation even over rather short time horizons, because the results are almost as good, and computing the optimal processor speeds is faster for (VI) than for (DP). This result is rather intuitive because the only important difference between (DP) and (VI) concerns the last steps. Indeed, during these last steps, (VI) behaves as if jobs will continue to arrive in the future (after $T - \Delta$), whereas (DP) considers that there is no job arrival after $T - \Delta$. (DP) can therefore adapt the chosen speeds in the last steps, whereas (VI) cannot. Thanks to this, the energy consumption of (DP) during the last steps is, on average, better than that of (VI).

Finally, the red curve shows that the energy difference between (OA) and (DP) is almost constant, whatever the value of the horizon time T. The horizon time has a limited impact on the energy difference: As for (VI), (OA) does not take into account the finite time horizon (except on the last Δ steps). This is why the red curve is also decreasing with the time horizon, but very slightly. Data in Table 6 confirm the results obtained in Example 2 before, because whatever the considered time horizon, the gain of (DP) in comparison with (OA) ranges between 4% and 5.5%.

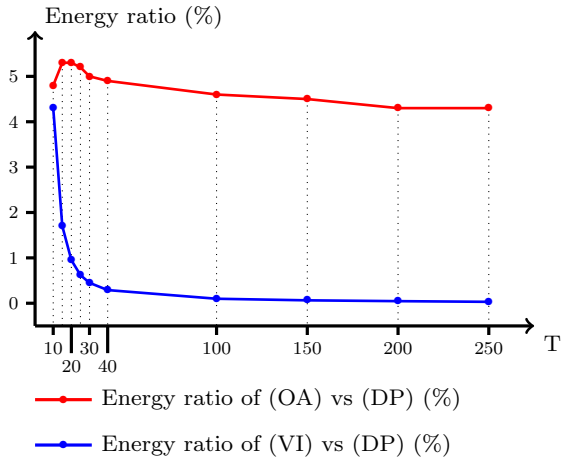

Fig. 6: Influence of the time horizon T on the energy difference between (OA) and (DP), and between (VI) and (DP), with Example 2.

5 Generalization of the Model

In the next three parts, we develop several extensions to make our model more realistic. To achieve this, we assume that the processor can change speeds at any time. This assumption is not very strong because there is no technical reason to change processor speed only at task arrival. These generalizations are the following:

1. Convexification of the power consumption function: Any non-convex power function

can be advantageously replaced by its convex hull.
2. Taking into account the time penalty required to change the processor speed: This time penalty can be replaced by an additional cost on the energy consumption.
3. Taking into account the context switching time between one task to another: This switching time can also be included in the cost function.

5.1 Convexification of the Power Consumption Function

Our general approach does not make any assumption on the power function $j(\cdot)$. Our algorithms (DP) and (VI) will compute the optimal speed selection for any function $j(\cdot)$. However structural properties (including the comparison with (OA) and the monotonicity) require the convexity assumption. It is therefore desirable to convexify the power function.

Let us consider a processor, whose speeds belong to the set $\mathcal{S} = \{s_0, s_1, s_2, s_{\max}\}$ and the power function of the processor $j(\cdot) : \mathcal{S} \to \mathbb{R}$. If

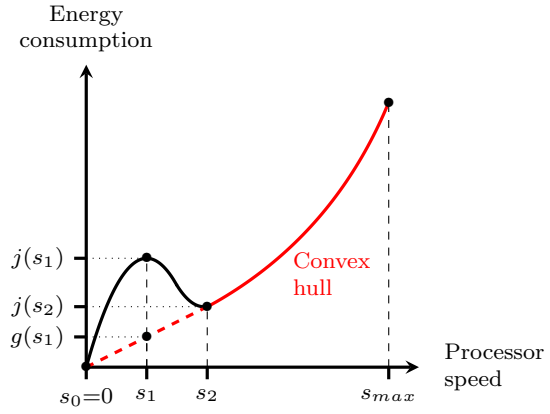

Fig. 7: Convexification of the power consumption function.

the power function is not convex, some speeds are not relevant, because using these speeds is more expensive in term of energy than using a combination of other speeds. Figure 7 depicts a non-convex power function j in black, and its convex hull g in red. In terms of energy consumption, it is better to choose speeds s_0 and s_2 (actually a linear combination of s_0 and s_2), rather than speed s_1. In fact, all points of the power function curve, that are above the convex hull, should never be taken into consideration. It is always better to only select the speeds whose power consumption belongs to the convex hull of the power function. Indeed if $g(s_1) < j(s_1)$ (see Figure 7), instead of selecting speed s_1 during any time interval $[t, t+1)$, the processor can select speed s_2 during a fraction of time α_2, and then speed s_0 during a fraction of time α_0, such that $\alpha_0 s_0 + \alpha_2 s_2 = s_1$. The total quantity of work executed during the time interval $[t, t+1)$ will be the same as with s_1, but the energy consumption will instead be $g(s_1) = \alpha_0 j(s_0) + \alpha_2 j(s_2)$, which is less than $j(s_1)$ because of the convexity of function j. This approach uses the V_{dd} hopping technique.

As a result, we can always consider that the power function is convex. This is very useful in practice. Indeed, the actual power consumption of a CMOS circuit working at speed s is non-convex function of the form $j(s) = Cs^\alpha + L(s)$, where the constant C depends on the activation of the logical gates, α is between 2 and 3, and $L(s)$ is the leakage, with $L(0) = 0$ and $L(s) \neq 0$ if $s > 0$. In this case, convexification removes the small values of s from the set of useful speeds.

Remark: This idea of replacing one speed by a linear combination of two speeds (*i.e.*, V_{dd} hopping) can also be used to simulate any speed between 0 and $s_{\max}$. Indeed, if a speed doesn't exist in the set $\mathcal{S}$, a solution is to simulate it by combining two neighboring speeds. This technique allows the processor to have more speeds to choose from, so that the optimal speed computed by the (DP) algorithm will use less energy with V_{dd} hopping than without it.

5.2 Taking into Account the Cost of Speed Changes

In our initial model, we have assumed that the time needed by the processor to change speeds is null. However, in all synchronous CMOS circuits, changing speeds does consume time and energy. The energy cost comes from the voltage regulator when switching voltage, while the time cost comes from the relocking of the Phase-Locked Loop when switching the frequency [24]. Burd and Brodersen have provided in [9] the equations to compute these two costs. In contrast with many DVFS studies (*e.g.*, [9,2,16,23]), our formulation can accommodate arbitrary energy cost to switch from speed s to s'. In the sequel, we denote this energy cost by $h_e(s, s')$.

As for the time cost, we denote by δ the time needed by the processor to change speeds. For the sake of simplicity we assume that the delay δ is the same for each pair of frequencies, but our formalization can accommodate different values of δ, as computed in [9]. During this time, the circuit logical functions are altered so no computation can take place.

With time delays for speed changes, the executed work by the processor has *two* slope changes, at times t_1 and t_2, with $t_2 - t_1 = \delta$ (see the red solid line in Figure 8). The problem is that, since in general $\delta \notin \mathbb{N}$, we cannot have both $t_1 \in \mathbb{N}$ and $t_2 \in \mathbb{N}$. As a consequence, one of the remaining work functions w_{t_1} or w_{t_2} of the state states $\mathbf{x}_{t_1}$ or $\mathbf{x}_{t_2}$ will not be integer valued. This is not allowed by our MDP approach.

We propose an original solution that replaces the actual behavior of the processor (represented by the red solid line in Figure 8) by a *simulated* behavior, equivalent in terms of the amount of work performed (represented by the blue dashed line in Figure 8). This simulated behavior exhibits a *single* speed change and is such that the total amount of work done by the processor is identical in both cases at all integer times (*i.e.* at $t_3 - 1$, t_3, and $t_3 + 1$ in Figure 8). The advantage is that, since there is only one state change, it can be chosen to occur at an integer time. In other words, we *choose* t_1 such that $t_3 \in \mathbb{N}$.

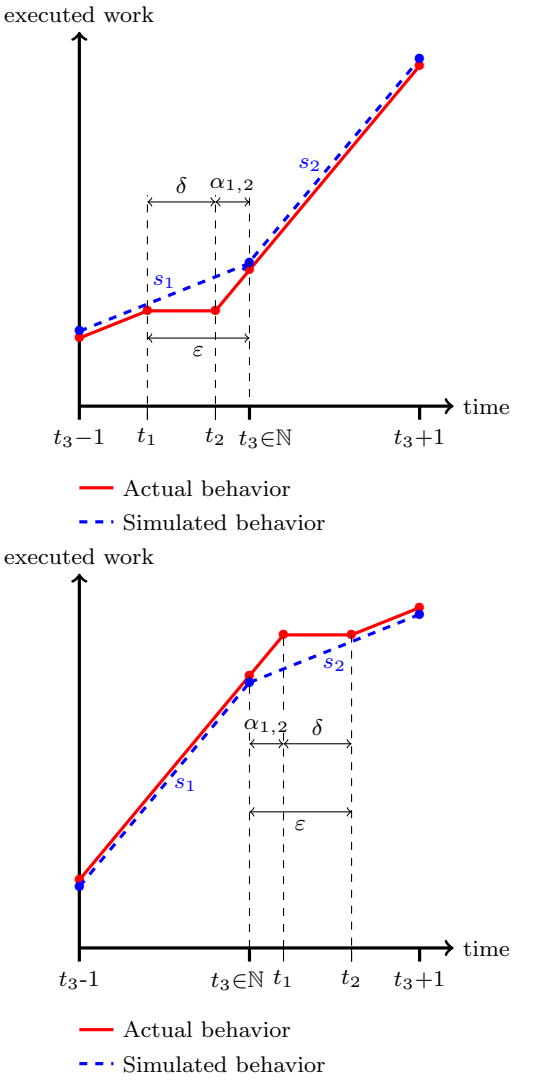

Fig. 8: Transformation of the time delay into an energy additional cost by shifting the switching point. The upper figure corresponds to the $s_1 < s_2$ case and the lower figure to the $s_1 > s_2$ case. The red line represents the actual behavior of the processor with a δ time delay. The blue dashed line represents an equivalent behavior in terms of executed work, with no time delay.

One issue remains, due to the fact that the consumed energy will not be identical with the real behavior and the simulated behavior; it

will actually be higher for the real behavior for convexity reason. This additional energy cost of the real processor behavior must therefore be added to the energy cost of the equivalent simulated behavior.

Finally, in order to trigger the speed change at time t_1, the processor needs to be "clairvoyant", *i.e.*, it needs to know in advance (before t_1) the characteristics of the job arriving at time t_3. This will allow the processor to compute the new speed s_2 and the length ε of the required interval to make sure that the work done by the processor at t_3 in the two cases (real and simulated) is identical.

The value of ε, $\alpha_{1,2}$, and the additional energy cost $h_\delta(s_1, s_2)$ of this speed change are computed as follows. In the case $s_2 > s_1$ (as in Figure 8), we have:

$$\begin{aligned} & s_1\varepsilon = s_2\alpha_{1,2} = s_2(\varepsilon - \delta) \\ \iff & (s_2 - s_1)\varepsilon = \delta s_2 \\ \iff & \varepsilon = \delta + \alpha_{1,2} = \frac{\delta s_2}{s_2 - s_1} \end{aligned} \tag{27}$$

We further assume that, during the time delay δ, the energy is consumed by the processor as if the speed were s_1. The additional energy cost incurred in the real behavior (red curve) compared with the simulated behavior (blue curve), denoted $h_\delta(s_1, s_2)$, is therefore:

$$h_\delta(s_1, s_2) = \alpha_{1,2}(j(s_2) - j(s_1))$$

Using the value of $\alpha_{1,2}$ from Eq. (27) yields:

$$h_\delta(s_1, s_2) = \delta s_1 \left(\frac{j(s_2) - j(s_1)}{s_2 - s_1} \right) \tag{28}$$

When $s_1 > s_2$, the additional cost becomes:

$$h_\delta(s_1, s_2) = \delta s_2 \left(\frac{j(s_1) - j(s_2)}{s_1 - s_2} \right) \tag{29}$$

This additional energy due to speed changes will be taken in consideration in our model in the cost function, by modifying the state space $\mathcal{X}$ and adding the current speed to the state at $t-1$. Therefore the new state at time t becomes $(\mathbf{x}_t, s_{t-1})$.

Taking into account both the energy cost and the time cost, the new main step of the (DP) Algorithm 1 becomes:

$$J^*_{t-1}(\mathbf{x}, s) \leftarrow \min_{s' \in \mathcal{A}(\mathbf{x})} \Big(j(s) + h_\delta(s, s') + h_e(s, s') + \sum_{\mathbf{x}' \in \mathcal{W}} P_t(\mathbf{x}, s', \mathbf{x}') J^*_t(\mathbf{x}', s') \Big) \tag{30}$$

with $h_\delta(s, s') = 0$ when $s = s'$, and otherwise given by Eq. (29) if $s' < s$ and Eq. (28) if $s < s'$. The rest of the analysis is unchanged.

5.3 Taking into Account the Cost of Context Switches

In the core of our article, we have neglected the context switch delay in EDF incurred by a preemption. This cost is orders of magnitude less than the cost of executing a job [4,7,8]. Nevertheless, in the following, we present a solution where we take into account this context switch delay.

5.3.1 Without Processor Sharing

When the processor can only execute one job at a time, one can consider that switching from the execution of one job to another one takes some time delay, denoted γ. This is essentially the time needed to upload or download the content of the execution stack. During this context switch, no useful work is being executed. This time delay is assumed to be identical for the beginning of a new job or the resuming of a job after preemption.

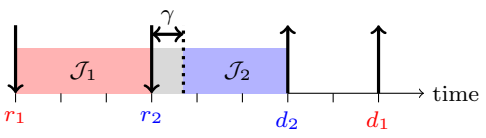

Fig. 9: Impact of a context switch on the execution time.

Figure 9 illustrates an example made of 2 jobs with the following characteristics: $\mathcal{J}_1(r_1 =$

$0, c_1 = 3, d_1 = 7, 5)$ and $\mathcal{J}_2(r_2 = 3, c_2 = 2, d_2 = 3)$. The switching time γ is marked with the barred area.

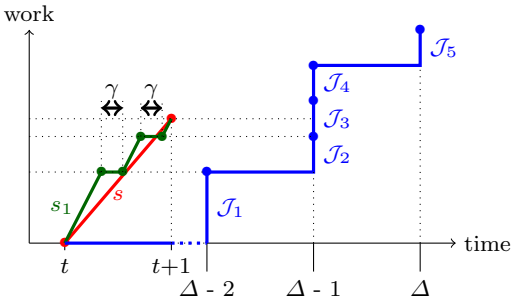

Fig. 10: Compensation of the impact of the context switches on the executed work by using a higher speed. The released jobs are $\mathcal{J}_i = (r_i, c_i, d_i)$, for $1 \leq i \leq 5$, with $r_5 + d_5 = \Delta$, $r_4 + d_4 = r_3 + d_3 = r_2 + d_2 = \Delta - 1$, and $r_1 + d_1 = \Delta - 2$.

Figure 10 illustrates the fact that, during one time step, several context switches can occur. In this example, during the time interval $[t, t+1)$, the processor completes two jobs, $\mathcal{J}_1$ and $\mathcal{J}_2$, and starts the execution of $\mathcal{J}_3$ (see the red curve). This involves two context switches, both of which occur during one time unit. This leads to a total delay of 2γ. As in Section 5.2, we transform this time delay into an energy cost: In one time unit, the evolution of the executed work under speed s_1, with K context switches (see the green curve), is the same as the evolution of the executed work under speed $s = s_1(1 - K\gamma)$, with no switching delay (red curve).

The state space of the system must be modified to be able to compute K, the number of context switches in each time interval. We must keep in memory the sizes of the jobs instead of only the total remaining work. Indeed, with the current state space $\mathcal{X}$, we do not know the number of actual different jobs composing a given amount $w(i), i \in \{0, \cdots, \Delta\}$, so we cannot know the number of context switches. We denote by $\overline{\mathcal{X}}$ the new state space and by $\overline{\mathbf{x}}_t \in \overline{\mathcal{X}}$ the new current state at time t:

$$\overline{\mathbf{x}}_t = (\overline{w}_t, \ell_t) \tag{31}$$

where $\overline{w}_t$ has the following form:

$$\overline{w}_t = \Big[(\rho_1^1, \cdots, \rho_1^{k_1}), \cdots, \underbrace{(\rho_i^1, \cdots, \rho_i^{k_i})}_{\substack{\text{remaining work quantity of jobs} \\ \text{with absolute deadline } t+i}}, \cdots, (\rho_\Delta^1, \cdots, \rho_\Delta^{k_\Delta})\Big] \tag{32}$$

where k_i is the number of jobs whose relative deadline is i time units away (hence their absolute deadline is $t + i$), and ρ_i^j is the work quantity of the j-st such such job.

For the sake of simplicity, we will consider that there is only one new job at t (instead of a set of jobs). The general case with multiple arrival will be identical, up to an increase of the state space.

To simplify we consider a single new arrival (τ_n, c_n, d_n) at time $t = r_n$ (recall that r_n is computed from the τ_k values with Eq. (1)). The case with several arrivals is a direct adaptation of the following formula.

If the processor speed at time $t-1$ is s_{t-1}, then at time t the next state $\overline{\mathbf{x}}_{t+1}$ becomes $(\overline{w}_{t+1}, \ell_{t+1})$, where $\overline{w}_{t+1}$ is:

$$\begin{aligned}
\overline{w}_{t+1} = \Big[&\Big((\rho_2^1 - f(1,1))^+, \cdots, \\
&(\rho_2^{k_1} - f(1, k-1))^+\Big), \\
&\cdots, \\
&\Big((\rho_{d_n}^1 - f(d_n, 1))^+, \cdots, \\
&(\rho_{d_n}^{k_{d_n}} - f(d_n, k_{d_n}))^+, c_n\Big), \\
&\cdots, \\
&\Big((\rho_\Delta^1 - f(\Delta, 1))^+, \cdots, \\
&(\rho_\Delta^{k_\Delta} - f(\Delta, k_\Delta))^+\Big), \\
&\Big((\rho_\Delta^1 - f(\Delta, 1))^+, \cdots, \\
&(\rho_\Delta^{k_\Delta} - f(\Delta, k_\Delta))^+\Big)\Big]
\end{aligned} \tag{33}$$

where

$$f(d,k) = \left(s_{t-1} - \sum_{i=1}^{d-1}\sum_{j=1}^{k_i} \rho_i^j - \sum_{j=1}^{k} \rho_d^j\right)^+$$

The idea of the state change is to set all the ρ_i^j values to 0 when $s \geq \sum_{i,j} \rho_i^j$. One job is executed partially and the others remain unchanged.

We further assume that the energy consumption during a context switch is the same as when some work is executed. The new main step of the Algorithm 1 for (DP) now has the following form:

$$J^*_{t-1}(\overline{\mathbf{x}}) \leftarrow \min_{s \in \mathcal{A}(\overline{\mathbf{x}})} \left(j\left(\tfrac{s}{1-K_s\gamma}\right) + \sum_{\overline{\mathbf{x}}' \in \overline{\mathcal{X}}} P_t(\overline{\mathbf{x}}, s, \overline{\mathbf{x}}') J^*_t(\overline{\mathbf{x}}') \right) \quad (34)$$

Note that the speed $\frac{s}{1-K_s\gamma}$ may not be directly available, but using the remark made in Section 5.1, one can easily simulate this speed with the neighboring available speeds.

Let K_s be the number of job executed if we use the speed s. We have:

$$K_s = \sum_{i=1}^{\alpha-1} k_i + \alpha \quad (35)$$

where $\alpha = \arg\min_{\alpha,k_\beta} \left(s < \sum_{i,j}^{\alpha,k_\beta} \rho_i^j \right)$. The rest of the analysis is unchanged.

5.3.2 With Processor Sharing

If processor sharing is enabled, which is often the case nowadays, the switching time is replaced by the additional delay *per time unit* caused by the permanent context switch. This additional delay is also denoted γ in this section.

In this case, the state space can be simplified. One only needs to keep in memory the number of jobs that are executed in a specific $w_t(i)$ from state $\mathbf{x}_t$, instead of all their sizes. Therefore, the state becomes

$$\mathbf{x}'_t = [w_t(1), \ldots, w_t(\Delta), k_t(1), \ldots, k_t(\Delta), \ell_t]$$

where $k_t(i)$ is the number of jobs with relative deadlines i. We have also to modify the change state function accordingly. Again we consider a single arrival (r_n, c_n, d_n) at time $t+1 = r_n$ (the general case is a direct extension). If the processor speed at time t is s_t, then at time $t+1$ the next state $\mathbf{x}'_{t+1} = (w_{t+1}, k_{t+1}, \ell_{t+1})$ is such that:

$$w_{t+1}(\cdot) = \mathbb{T}\left[(w_t(\cdot) - s_t)^+\right] + c_n H_{d_n}$$

as before, ℓ_{t+1} also follows the same evolution as in the original case, and for all $i = 1 \ldots \Delta$,

$$k_{t+1}(i) = \begin{cases} \mathbf{1}_{\{i=d_n\}} & \text{if } \; s_t > w_t(i+1) \\ k_t(i+1) + \mathbf{1}_{\{i=d_n\}} & \text{otherwise,} \end{cases}$$

In the processor sharing case, the additional time due to switching is γ per time unit. The Bellman equation is the same as Eq. (34) but replacing K_s by:

$$K'_s = \sum_i \frac{\mathbf{1}_{\{k_i>1\}}}{s} \left(s - w(i-1)\right)^+ + \mathbf{1}_{\{k_1>1\}} \frac{w(1)}{s} \quad (36)$$

6 Conclusion and Perspectives

In this paper, we showed how to select on-line speeds to execute real-time jobs while minimizing the energy consumption by taking into account statistic information on job features. This information may be collected by using past experiments or simulations, as well as deductions from the structure of job sources. Our solution provides performances that are close to the optimal off-line solutions on average, and outperforms classical on-line solutions in cases where the job features have distributions with large variances.

While the goal of this study is to propose a better processor speed policy, several points are still open and will be the topic of future investigations.

The first one concerns the scheduling model: In this paper jobs are executed under the *Earliest Deadline First* policy, but this is not always possible in practice. What would be the consequence of using another scheduling policy?

The second one concerns the time and space complexity of our algorithms. These complexities are exponential in the deadlines of the

jobs. Although our algorithms (DP) and (VI) are used off-line and can be run on powerful computers, our approach remains limited to a small range of parameters. One potential solution is to simplify the state space and to aim for a sub-optimal solution (but with proven guarantees), using approximate dynamic programming.

Finally, the statistical information gathered on the job features is crucial. When this information is not accurate or even not available, Markov Decision approaches are not possible and one should use reinforcement learning techniques (such as Q-learning [5]) to construct a statistical model of the jobs on-line and select the speeds accordingly, which will converge towards optimal speeds over time.

Appendix: Size of the State Space

This appendix is dedicated to the enumeration of the total number of states (Eqs. (10), (11), and (12)).

Let $w(\cdot)$ be a valid state of the system, at any time t. Since all parameters are integer numbers and the maximum deadline of a task is Δ, the maximal look-ahead at any time is Δ, hence $w(\cdot)$ is characterized by its first Δ integer values (that are non-decreasing by definition): $w(1) \leq \cdots \leq w(\Delta)$.

Let us define the step sizes of w, starting from the end, as: $z_1 = w(\Delta) - w(\Delta - 1)$, and more generally, $z_j = w(\Delta - j + 1) - w(\Delta - j)$, for all $j = 1, \ldots, \Delta$.

The arrived work at any time t being of maximal size C, we have $z_1 \leq C$ because z_1 must be bounded by the amount of work that was released at time t. Similarly, $z_1 + z_2$ must be bounded by the amount of work that has arrived at times t and $t-1$, namely $2C$, and so on and so forth up to $z_1 + z_2 + \cdots + z_\Delta \leq \Delta C$. This is the only condition for a function w to be a possible state when deadlines and sizes are arbitrary integers bounded by Δ and C respectively.

Therefore $(z_1, z_2, \ldots, z_\Delta)$ satisfies the following conditions:

$$\begin{cases} z_1 \leq C \\ z_1 + z_2 \leq 2C \\ z_1 + z_2 + z_3 \leq 3C \\ \vdots \\ z_1 + z_2 + \ldots + z_\Delta \leq \Delta C \end{cases}$$

By defining the partial sums $y_j = z_1 + \cdots + z_j$, the number of states satisfies:

$$Q(\Delta, C) = \sum_{y_1=0}^{C} \sum_{y_2=y_1}^{2C} \sum_{y_3=y_2}^{3C} \cdots \sum_{y_\Delta=y_{\Delta-1}}^{\Delta C} 1$$

This multiple sum can be seen as a generalized Catalan number. Indeed, a state characterized by its steps $(z_1, \ldots, z_\Delta)$ is in bijection with a path on the integer grid from $(0,0)$ to $(\Delta+1, C(\Delta+1))$, which remains *below* the diagonal of slope C (see Fig. 11).

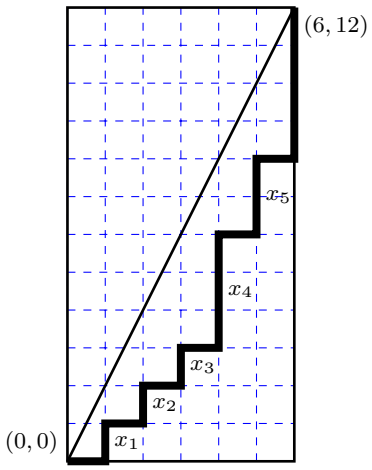

Fig. 11: A valid state seen as a path below the diagonal $(0,0) - (6,12)$, for $C = 2$ and $\Delta = 5$.

Counting the number of such paths has been done in [14] and corresponds to the generalized Catalan numbers, $C_n^k = \frac{1}{nk+1}\binom{nk+1}{n}$, with $k = C+1$ and $n = \Delta+1$. Therefore,

$$Q(\Delta, C) = \frac{1}{1 + C(\Delta+1)} \binom{(C+1)(\Delta+1)}{\Delta+1}$$

Using the Stirling formula, we finally get:

$$Q(\Delta, C) \approx \frac{e}{\sqrt{2\pi}} \frac{1}{(\Delta+1)^{3/2}} (e\,C)^\Delta$$

References

1. Aydin, H., Melhem, R.G., Mossé, D., Mejía-Alvarez, P.: Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In: Euromicro Conference on Real-Time Systems, ECRTS'01, pp. 225–232. IEEE Computer Society, Delft, The Netherlands (2001)
2. Bandari, M., Simon, R., Aydin, H.: Energy management of embedded wireless systems through voltage and modulation scaling under probabilistic workloads. In: International Green Computing Conference, IGCC'14, pp. 1–10. IEEE Computer Society, Dallas, TX, USA (2014)
3. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. Journal of the ACM **54**(1) (2007)
4. Bastoni, A., Brandenburg, B., Anderson, J.: Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In: Workshop on Operating Systems Platforms for Embedded Real-Time Applications, OSPERT'10, pp. 33–44 (2010)
5. Bertsekas, D., Tsitsiklis, J.: Neuro-dynamic programming. Athena Scientific, Belmont, Mass., USA (1996)
6. Bini, E., Scordino, C.: Optimal two-level speed assignment for real-time systems. IJES **4**(2), 101–111 (2009)
7. Brandenburg, B.: Scheduling and locking in multiprocessor real-time operating systems. Ph.D. thesis, The University of North Carolina at Chapel Hill, Chapel Hill (NC), USA (2011)
8. Brandenburg, B., Gul, M.: Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations. In: Real-Time Systems Symposium, RTSS'16, pp. 99–110. IEEE Computer Society (2016)
9. Burd, T., Brodersen, R.: Design issues for dynamic voltage scaling. In: International Symposium on Low Power Electronics and Design, ISLPED'00. Rapallo, Italy (2000)
10. Gaujal, B., Girault, A., Plassart, S.: Dynamic speed scaling minimizing expected energy consumption for real-time tasks. Tech. Rep. hal-01615835, Inria (2017)
11. Gaujal, B., Girault, A., Plassart, S.: A Discrete Time Markov Decision Process for Energy Minimization Under Deadline Constraints. Research report, Grenoble Alpes ; Inria Grenoble Rhône-Alpes, Université de Grenoble (2019). URL https://hal.inria.fr/hal-02391948
12. Gaujal, B., Navet, N., Walsh, C.: Shortest Path Algorithms for Real-Time Scheduling of FIFO tasks with Minimal Energy Use. ACM Transactions on Embedded Computing Systems (TECS) **4**(4) (2005)
13. Gruian, F.: On energy reduction in hard real-time systems containing tasks with stochastic execution times. In: IEEE Workshop on Power Management for Real-Time and Embedded Systems, pp. 11–16 (2001)
14. Hilton, P., Pedersen, J.: Catalan numbers, their generalization, and their uses. The Mathematical Intelligencer **13**(2), 64–75 (1991)
15. Horn, W.: Some simple scheduling algorithms. Naval Research Logistics **21**(1), 177–185 (1974)
16. Li, K.: Energy and time constrained task scheduling on multiprocessor computers with discrete speed levels. J. of Parallel and Distributed Computing **95**(C), 15–28 (2016)
17. Lorch, J., Smith, A.: Improving dynamic voltage scaling algorithms with PACE. In: ACM SIGMETRICS 2001 Conference, pp. 50–61 (2001)
18. Mao, J., Cassandras, C.G., Zhao, Q.: Optimal dynamic voltage scaling in energy-limited nonpreemptive systems with real-time constraints. IEEE Trans. Mob. Comput. **6**(6), 678–688 (2007). DOI 10.1109/TMC.2007.1024. URL https://doi.org/10.1109/TMC.2007.1024
19. Marshall, A., Olkin, I.: Inequalitites: Theory of Majorization and Its Applications, *Mathematics in Science and Engineering*, vol. 143. Academic Press (1979)
20. Müller, A., Stoyan, D.: Comparison Methods for Stochastic Models and Risks. No. ISBN: 978-0-471-49446-1 in Wiley Series in Probability and Statistics. Wiley (2002)
21. Pillai, P., Shin, K.G.: Real-time dynamic voltage scaling for low-power embedded operating systems. SIGOPS Oper. Syst. Rev. **35**(5), 89–102 (2001)
22. Puterman, M.L.: Markov Decision Process : Discrete Stochastic Dynamic Programming, wiley series in probability and statistics edn. Wiley (2005)
23. Wang, J., Roop, P., Girault, A.: Energy and timing aware synchronous programming. In: International Conference on Embedded Software, EMSOFT'16. Pittsburgh (PA), USA (2016). DOI 10.1145/2968478.2968500. URL https://hal.inria.fr/hal-01412100
24. Wu, Q., Juang, P., Martonosi, M., Clark, D.: Voltage and frequency control with adaptive reaction time in multiple-clock-domain processors. In: International Conference on High-Performance Computer Architecture, HPCA'05, pp. 178–189. IEEE, San Francisco (CA), USA (2005)
25. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proceedings of IEEE Annual Foundations of Computer Science, pp. 374–382 (1995)