# Exploiting Job Variability to Minimize Energy Consumption under Real-Time Constraints

Bruno Gaujal, Girault Alain, Stéphan Plassart


# Exploiting Job Variability to Minimize Energy Consumption under Real-Time Constraints

**Bruno Gaujal, Girault Alain, Stéphan Plassart**

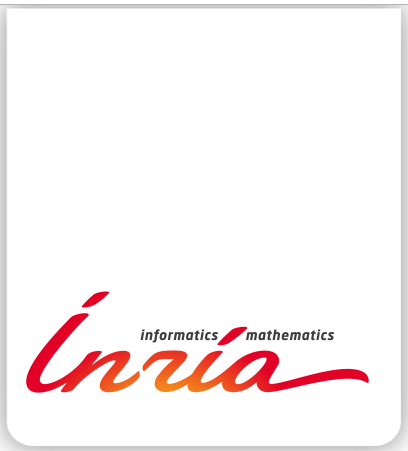
**RESEARCH REPORT**

**N° 9300**

November 2019

Project-Teams Polaris and Spades

ISSN 0249-6399 ISRN INRIA/RR--9300--FR+ENG




# Exploiting Job Variability to Minimize Energy Consumption under Real-Time Constraints

Bruno Gaujal, Girault Alain, Stéphan Plassart *

Project-Teams Polaris and Spades

Research Report n° 9300 — November 2019 — 23 pages

**Abstract:** This paper proposes a Markov Decision Process (MDP) approach to compute the optimal on-line speed scaling policy that minimizes the energy consumption of a single processor executing a finite or infinite set of jobs with real-time constraints, in the non-clairvoyant case, *i.e.*, when the actual execution time of the jobs is unknown when they are released. In real life applications, it is common at release time to know only the Worst-Case Execution Time of a job, and the *actual* execution time of this job is only discovered when it finishes. Choosing the processor speed purely in function of the Worst-Case Execution Time is sub-optimal. When the probability distribution of the actual execution time is known, it is possible to exploit this knowledge to choose a lower processor speed so as to minimize the expected energy consumption (while still guaranteeing that all jobs meet their deadline). Our MDP solution solves this problem optimally with discrete processor speeds. Compared with approaches from the literature, the gain offered by the new policy ranges from a few percent when the variability of job characteristics is small, to more than 50% when the job execution time distributions are far from their worst case.

**Key-words:** Optimization, Real-Time Systems, Markov Decision Process, Dynamic Voltage and Frequency Scaling.

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

**RESEARCH CENTRE**
**GRENOBLE – RHÔNE-ALPES**

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

# Exploitation de la variabilité des tâches pour minimiser la consommation d'energie sous des contraintes temps-réel

**Résumé :** Ce papier propose une approche de processus à décision de Markov (PDM) pour calculer la politique de vitesses en-ligne optimale qui minimise la consommation d'énergie d'un seul processeur qui exécute un ensemble de jobs fini ou infini avec des contraintes temps-réel, dans le cas non-clairvoyant, *i.e.*, quand le temps d'execution réel des tâches est inconnu lorsqu'elles arrivent. Quand la probabilité de distribution du temps d'execution réel est connu, il est possible d'exploiter cette connaissance pour choisir une vitesse de processeur plus faible afin de minimiser la consommation d'énergie espérée (tout en garantissant que les tâches finissent avant leurs deadlines).

**Mots-clés :** Optimisation, Système temps-réel, Processus à Décision de Markov, Adaptation en Fréquence et Ajustement Dynamique de Tension.

# Contents



# 1 Introduction

The *single processor hard real-time energy minimization problem* consists in choosing, at each decision time, the clock frequency of the processor to execute the current jobs, such that all jobs meet their deadline and such that the total energy consumed by the processor is minimized.

Hard real time constraints and energy minimization are difficult to combine because the former require to be very conservative by only considering the worst cases, while the latter would benefit greatly from relaxing strict deadlines for job completions. Nevertheless, several approaches have been proposed to tackle the hard real-time energy minimization problem under several assumption on the processor and on the jobs to be executed.

The most classical case is the *off-line case*, where the number of jobs is finite and all their characteristics are known before choosing the processor speeds. The finite set is jobs is $\{J_i\}_{i\in\{1,..,N\}}$, each one defined by its *release time* $r_i$, its *execution time* $c_i$ (at the nominal processor speed), and its *relative deadline* $d_i$. Based on this complete knowledge, the processor must select, at each time $t$ in $[0, r_N]$, its operating speed $s(t)$ among a given set of admissible speeds $\mathcal{S}$, such that all jobs are completed before their deadline *and* the total energy consumption of the processor over the interval $[0, r_N + d_N]$ (a function of its speed) is minimized. The off-line case has been first introduced by Yao et al. in [1], which proposed the (YDS) off-line greedy algorithm, and then solved optimally by Gaujal et al. in [2] in the FIFO case.

The *on-line case* differs in that, at time $t$, only the jobs released before $t$ or at $t$ are known, and the number of jobs can be infinite. We further distinguish the *clairvoyant on-line case*, where the characteristics of each job $J_i$ (its execution time $c_i$ and its deadline $d_i$) are revealed at the release time of $J_i$ (*i.e.*, at $r_i$). This case has been first investigated by Yao et al. who proposed the Optimal Available (OA) — a greedy speed policy — and the Average Rate (AVR) — a proportional fair speed policy [1]. These speed policies have been compared with the optimal off-line solution by Bansal et al. in [3], who also computed their competitive ratio. Further improvements have been proposed in [4] and [3].

In the *non-clairvoyant on-line case*, for each job $J_i$ only its deadline and its Worst Case Execution Time (WCET) are revealed to the processor when it is released, but the *actual execution time* of $J_i$ is only known when it finishes. This case has been first investigated by Lorch and Smith in [5,6] for a *single job*: they have proposed the Processor Acceleration to Conserve Energy approach (PACE), which provides an analytic formula that allows to compute the continuous evolution of the processor speed during the life time of the job.

To the best of our knowledge, (PACE) has been extended in three directions. First, Xu et al. have studied the practical case with discrete speeds, no assumption on the power function, non-null processor idle power, and non-null speed switching overhead [7]. The authors have proposed Practical PACE (PPACE), a fully polynomial time approximation scheme $\varepsilon$-optimal algorithm in the single job case, which performs a time discretization of the speed selection. Second, Bini and Scordino have proposed an optimal solution to the particular case where the processor uses only two speeds to execute the job, taking into account the speed switching overhead [8]. Third, Zhang et al. have proposed the Optimal Procrastinating Dynamic Voltage Scaling algorithm (OPDVS) under the form of a constrained optimization problem [9].

Another series of papers have relaxed the single job assumption of (PACE). Considering several jobs instead of a single job gives rise to the distinction between *sporadic* and *periodic* jobs. In the periodic case, several papers have focused on the constrained framework of a *frame based multi-task model* [10,11], where all the tasks are periodic, with their deadline equal to their period, and share the *same period*. In this context, Zhang et al. have proposed the Global Optimal Procrastinating Dynamic Voltage Scaling algorithm (OPDVS) [9], while Xu et al. have proposed a Hybrid Dynamic Voltage Scaling algorithm (HDVS), hybrid in the sense that it addresses both intra-task Dynamic Voltage Scaling (DVS) and inter-task DVS [12]. The (HDVS) algorithm is a fully polynomial time approximation scheme $\varepsilon$-optimal algorithm.

The drawback is that the frame based model can be restrictive. Indeed, modern real-time systems exhibit a combination of *sporadic* tasks and of periodic tasks with *significantly different* periods (typically ranging between $1\,ms$ and $1,000\,ms$). The former cannot be captured at all in a frame based model, and for the latter, a decomposition of the hyper-period schedule into frames would result in too many frames to be practical, and more importantly would be sub-optimal in terms of energy consumption.

Finally, Lorch and Smith also proposed a multi-job extension of (PACE) in [6] by considering each job *independently* and adding the speeds obtained for each job in isolation, resulting in a sub-optimal speed selection.

The goal of this paper is to find an optimal solution to the non-clairvoyant on-line problem. We build a Markov Decision Process (MDP) that computes the *optimal* speed of the processor at each time instant, in order to minimize the expected energy consumption while guaranteeing the completion of all jobs before their deadline. To achieve this, we design a finite state space for the evolution of the system and compute the transition probabilities from one state to another, based on the distributions of the job characteristics. The combinatorial cost of this construction is significant, but since the computations of the transition probabilities and of the optimal speed policy is done off-line, we claim it does not hamper the on-line usability of the resulting speed

policy for an embedded system.

We also run several numerical experiment to assess the gain over sub-optimal solutions, such as the superposition of [6].

The paper is organized as follows. Related work having already been covered in the introduction, we formalize the problem in Sec. 2. Then we build our MDP solution in Sec. 3. We compare our solutions with previous work in Sec. 4. We perform numerical experimentation on synthetic and real-life benchmarks in Sec. 5. Finally we give concluding remarks in Sec. 6.

# 2 Formalization

## 2.1 System Model

Each job $J_i$ is defined by the triplet $(\tau_i, c_i, d_i)$, where $\tau_i$ is the *inter-arrival time* between $J_i$ and $J_{i-1}$. The inter-arrival times are bounded by $L$. From the $\tau_i$ values, we can reconstruct the *release time* $r_i$ of each job $J_i$ as follows: we fix $r_0 = 0$ by convention, and then for all $i > 0$, we take $r_i = r_{i-1} + \tau_i$. We take the inter-arrival time as an intrinsic parameter of the jobs because it allows us to define it as a random variable with a probability distribution, which would not be possible with the release times since they go to $+\infty$.

The two others parameters are the *execution time* $c_i$, bounded by $C$, and the *relative deadline* $d_i$, bounded by $\Delta$. We assume that all these quantities are in $\mathbb{N}$. If the actual values are rational numbers, a multiplicative rescaling is used to make them all integer.

The single processor is equipped with DVFS capability and is characterized by a finite set of available speeds, also in $\mathbb{N}$: $\mathcal{S} = \{s_1 = 0, s_2, \ldots, s_k = s_{\max}\}$. For any job $J_i$, the execution time $c_i$ is the execution time at the nominal speed 1 (the slowest possible speed).

Processor speed changes may occur only at integer times. The cost of speed switching is assumed to be null for simplicity. Our approach can be generalized with a non-null speed switching cost by modifying the cost function in the MDP.

The dynamic evolution of the system is as follows. It starts at time 0 with an empty state. The first job $J_1 = (\tau_1, c_1, d_1)$ arrives at time $r_1 = \tau_1 > 0$. The processor uses one of its available speeds $s \in \mathcal{S}$ to start executing $J_1$. The next job arrives, and so on and so forth either until reaching the time horizon $T$ (equal to $r_N + d_N$), or forever in the case of an infinite number of jobs.

When several jobs are present in the system at a given time, some scheduling policy must be used. Since the on-line speed policy we propose in this paper is "global" (in the sense that the speed of the processor at time $t$ depends on the global state of the system, which itself depends on all the jobs active at time $t$), we use the EDF (Earliest Deadline First) scheduling policy. A key advantage of EDF is that it is optimal for *feasibility*. A set of jobs is *feasible* if no deadline is missed when the processor always uses its maximal speed $s_{\max}$. In this respect, EDF is optimal means that if a set of jobs is feasible for some scheduling policy $\Pi$, then it is also feasible for EDF.

In Sec. 4 we compare our speed policy with (OA) [1] and (PACE) [5]. (OA) is a global policy too, so we also use EDF when several jobs are active at the same time. In contrast, (PACE) is a "local" policy (in the sense that the speed of the processor is computed individually for each job, and then summed up over all the jobs active at time $t$), so in this case we use a processor sharing policy.

The power dissipated at any time $t$ by the processor running at speed $s(t)$ is denoted $P_{ower}(s(t))$. No assumption is made on the $P_{ower}$ function (unlike (OA) and (PACE) which

both assume that it is *convex*). In the finite case, we compute the total energy consumption as:

$$E = \sum_{t=0}^{T} P_{ower}(s(t)), \tag{1}$$

while in the infinite case, we compute the long run energy consumption *average* as:

$$g = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} P_{ower}(s(t)). \tag{2}$$

Our goal is to execute all the jobs before their deadline while minimizing the total or average energy consumption. In the following, we solve this constraint optimization problem on-line. At any time $t$, the processor does not know the future releases, nor the exact duration of currently executed jobs. Instead of investigating an adversarial model (worst possible future arrivals as well as job duration), we focus on a *statistical model*. The variables $\tau_i, c_i, d_i$ are viewed as *random variables*, for which we have *probability distributions*. Such distributions can be either given by the system designer or be estimated from numerous executions of the system.

## 2.2 State Space

The information available to the processor to choose its speed can be split in two parts. The static part consists of the distributions of the execution times, release times, and deadlines of the jobs. The dynamic part changes over time, it will be called the *system state* of the system in the following.

**Definition 1 (System state)** *The system state at time $t$, denoted $\boldsymbol{w}(t)$, is composed of two elements:*

- *$\ell$: the time elapsed since the latest job arrival.*
- *$\boldsymbol{x}$: the list of current jobs, sorted by their deadlines (ties are sorted by job release times); in this list, each job $J_i$ is represented by a pair $(e_i, d_i)$ where:*
  - *$e_i$ is the work quantity already executed by the processor on job $J_i$;*
  - *$d_i$ is the relative deadline of job $J_i$.*

In the following we denote by $\mathcal{W}$ the state space and by $\mathcal{X}$ the space of all possible lists $\boldsymbol{x}$ of pairs $(e_i, d_i)$. So $\mathcal{W} = \{1, \ldots, L\} \times \mathcal{X}$, where $L$ is the maximal inter-arrival time between any two jobs.

## 2.3 State Space Evolution

To analyze the evolution of the system state from time $t$ to $t+1$, we only focus on the space $\mathcal{X}$ (*i.e.*, all the possible lists of jobs) and on its evolution over time, because the evolution of $\ell$ is trivial.

In the following we simply put into formula the two possible changes of the state: (i) some jobs may be completed during the current time interval $(t, t+1]$ by the processor running at its current speed $s(t)$, which are removed from $\boldsymbol{x}(t)$; and (ii) some new jobs may arrive at time $t+1$, which must be inserted in the list $\boldsymbol{x}(t+1)$.

Two operators that will be used to formalize job completions and job arrivals.

**Definition 2** *Let* $\boldsymbol{x} = [(e_1^{\boldsymbol{x}}, d_1^{\boldsymbol{x}}), \ldots, (e_n^{\boldsymbol{x}}, d_n^{\boldsymbol{x}})]$ *and* $\boldsymbol{y} = [(e_1^{\boldsymbol{y}}, d_1^{\boldsymbol{y}}), \ldots, (e_n^{\boldsymbol{y}}, d_n^{\boldsymbol{y}})]$ *be two job lists. We define two binary operators:*

- $\boldsymbol{x} \oplus \boldsymbol{y}$ *returns the sorted union of the two job lists* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *(sorted by the jobs' deadlines).*
- $\boldsymbol{x} \ominus \boldsymbol{y}$ *returns the sorted list of jobs of* $\boldsymbol{x}$ *that are not in* $\boldsymbol{y}$ *(sorted by the jobs' deadlines). By definition,* $\boldsymbol{x} \subset \boldsymbol{y} \Rightarrow \boldsymbol{x} \ominus \boldsymbol{y} = \emptyset$.

Let $\boldsymbol{x}_t$ denote the list of jobs and $s(t)$ the speed used by the processor, both at time $t$. The job list $\boldsymbol{x}_t$ is denoted

$$\boldsymbol{x}_t = \left[(e_1^t, d_1^t), \ldots, (e_n^t, d_n^t)\right]. \tag{3}$$

Let us consider that the processor speed leads to the completion of $u$ jobs and a partial execution of the $(u+1)^{th}$ job. Then the remaining job list $\boldsymbol{x}_{t+1}$ contains the $(n-u)$ jobs that have not been totally executed by the processor before time $t+1$. So the next job list, $\boldsymbol{x}_{t+1}$, is composed of at least the following jobs:

$$\left[(e_{u+1}^{t+1}, d_{u+1}^{t+1}), \ldots, (e_n^{t+1}, d_n^{t+1})\right] \tag{4}$$

where $\forall k \in \{u+1, \ldots, n\}, d_k^{t+1} = d_i^t - 1$ for all the jobs present at $t$ and not finished at $t+1$ (their deadlines get closer), $e_{u+1}^{t+1}$ is the new total work amount executed on job $J_{u+1}$ so far, and $e_i^{t+1} = e_i^t$ for all $i > u+1$.

In the sequel, we introduce the operator $\mathit{Shift}_u$ that implements all theses modifications on the job list $\boldsymbol{x}_t$, before the new job arrivals, where $u$ is the number of jobs that are completed during the current time step and $r_\alpha$ is the work quantity executed on the unfinished $(u+1)^{th}$ job:

$$\mathit{Shift}_u(\boldsymbol{x}_t, r_\alpha) = \left[(e_{u+1}^t + r_\alpha, d_{u+1}^t - 1), (e_{u+2}^t, d_{u+2}^t - 1), \ldots, (e_n^t, d_n^t - 1)\right].$$

Next, we have to consider the jobs released at time $t+1$. The list of jobs released at time $t+1$, ordered by their deadlines, is denoted $a(t+1)$. Finally, the next job list $\boldsymbol{x}_{t+1}$ is:

$$\boldsymbol{x}_{t+1} = \mathit{Shift}_u(\boldsymbol{x}_t, r_\alpha) \oplus a(t+1).$$

## 2.4 Construction of the Transition Probability Matrix

In this section we construct the transition matrix $P_t((\boldsymbol{x}, \ell_{\boldsymbol{x}}), s, (\boldsymbol{y}, \ell_{\boldsymbol{y}}))$ that gives the probability to go from state $(\boldsymbol{x}, \ell_{\boldsymbol{x}})$ to state $(\boldsymbol{y}, \ell_{\boldsymbol{y}})$ over time step $(t, t+1]$, when the processor uses speed $s$.

In the following, we give an explicit construction of $P_t((\boldsymbol{x}, \ell_{\boldsymbol{x}}), s, (\boldsymbol{y}, \ell_{\boldsymbol{y}}))$ as a function of the distributions of the inter-arrival times, the sizes, and the deadlines of jobs, when jobs are i.i.d.[1]. In the general non-i.i.d. case, the matrix $P_t$ can still be constructed numerically, but no general closed form formula exists in this case.

The distributions of the job features are denoted as follows:

- The i.i.d. inter-arrival times have a common distribution denoted $\theta$:

$$\forall 0 \leq t \leq L,\ \theta(t) = \mathbb{P}(\tau_i = t) \text{ for any job } J_i.$$

- The i.i.d. size distribution is denoted $\sigma$:

$$\forall 1 \leq c \leq C,\ \sigma(c) = \mathbb{P}(c_i = c) \text{ for any job } J_i.$$

[1] i.i.d. = independent and identically distributed.

- The i.i.d. deadline distribution is denoted $\delta$:

$$\forall 1 \leq k \leq \Delta,\ \delta(k) = \mathbb{P}(d_i = k) \text{ for any job } J_i.$$

When all jobs are i.i.d., the transition probability $P_t((\boldsymbol{x}, \ell_{\boldsymbol{x}}), s, (\boldsymbol{y}, \ell_{\boldsymbol{y}}))$ does not depend on $t$ and can be decomposed in several parts as shown below:

$$P_t((\boldsymbol{x}, \ell_{\boldsymbol{x}}), s, (\boldsymbol{y}, \ell_{\boldsymbol{y}})) = \sum_{u=\lfloor \frac{s}{C} \rfloor}^{\min(s, n(\boldsymbol{x}))} P((\boldsymbol{x}, \ell_{\boldsymbol{x}}), u, s, (\boldsymbol{y}, \ell_{\boldsymbol{y}})). \tag{5}$$

where $n(\boldsymbol{x})$ is the number of jobs in the list $\boldsymbol{x}$, and $u$ the number of jobs completed during interval $(t, t+1]$.

The probability to go from state $(\boldsymbol{x}, \ell_{\boldsymbol{x}})$ to state $(\boldsymbol{y}, \ell_{\boldsymbol{y}})$ with $u$ jobs completed during a time step can be further decomposed into the probability $P_{exec}$ that $u$ jobs are executed at speed $s$ times the probability $P_a$ that a set $a(t)$ of jobs arrive in interval $(t, t+1]$:

- If $u = n(\boldsymbol{x})$, then $\forall i \in [1, u], 1 \leq k_i \leq C$:

$$P((\boldsymbol{x}, \ell_{\boldsymbol{x}}), u, s, (\boldsymbol{y}, \ell_{\boldsymbol{y}})) = \sum_{k_1 + \cdots + k_u \leq s} P_{exec}(\boldsymbol{x}, u, 0)\, P_a(\boldsymbol{x}, u, \boldsymbol{y}) \tag{6}$$

- Else if $u < n(\boldsymbol{x})$, then $\forall i \in [1, u], 1 \leq k_i \leq C$:

$$P((\boldsymbol{x}, \ell_{\boldsymbol{x}}), u, s, (\boldsymbol{y}, \ell_{\boldsymbol{y}})) = \sum_{k_1 + \cdots + k_u = s - (e_\alpha^{\boldsymbol{y}} - e_{u+1}^{\boldsymbol{x}})} P_{exec}(\boldsymbol{x}, u, e_\alpha^{\boldsymbol{y}})\, P_a(\boldsymbol{x}, u, \boldsymbol{y}) \tag{7}$$

where $e_\alpha^{\boldsymbol{y}}$ is the work quantity that corresponds to the first job of deadline $d_\alpha$ in state $\boldsymbol{y}$, $d_\alpha$ being the deadline of the first job of $\mathit{Shift}_u(\boldsymbol{x}, e_\alpha^{\boldsymbol{y}} - e_{u+1}^{\boldsymbol{x}})$. By definition, we have:

$$0 \leq e_{u+1}^{\boldsymbol{x}} \leq e_\alpha^{\boldsymbol{y}} \leq C - 1.$$

We now compute each term $P_{exec}$ and $P_a$:

▶ $P_{exec}$ is the probability that the $u$ first jobs of $\boldsymbol{x}$ are completed — the size of each of them depends (i) on the remaining work executed during $(t, t+1]$, $k_i$, $i \in [1, u]$ and (ii) on the amount of work $e_i^{\boldsymbol{x}}$ already executed before $t$ — times the probability that the job $u+1$ has been partially executed, given the work quantity already executed in the past (*i.e.*, before $t$). We distinguish two cases, $u \neq 0$ and $u = 0$.

The first case is $u \neq 0$:

$$\begin{aligned} P_{exec}(\boldsymbol{x}, u, e_\alpha^{\boldsymbol{y}}) &= \prod_{i=1}^{u} \mathbb{P}(V = k_i + e_i^{\boldsymbol{x}} \,|\, V > e_i^{\boldsymbol{x}})\ \mathbb{P}(V > e_\alpha^{\boldsymbol{y}} \,|\, V > e_{u+1}^{\boldsymbol{x}}) \\ &= \prod_{i=1}^{u} \theta(k_i + e_i^{\boldsymbol{x}} \,|\, V > e_i^{\boldsymbol{x}}) \left( \sum_{k=e_\alpha^{\boldsymbol{y}}}^{C} \theta(k) \Big/ \sum_{k=e_{u+1}^{\boldsymbol{x}}}^{C} \theta(k) \right) \end{aligned} \tag{8}$$

And the second case is $u = 0$:

$$P_{exec}(\boldsymbol{x}, 0, e_\alpha^{\boldsymbol{y}}) = 1. \tag{9}$$

▶ $P_a(\boldsymbol{x}, u, \boldsymbol{y})$ is the probability related to the new jobs arrivals. The computation of $P_a$ depends on the new jobs arrival $a(t)$, which is formally defined as:

$$a(t) = \boldsymbol{y} \ominus \mathit{Shift}_u(\boldsymbol{x}, e_\alpha^{\boldsymbol{y}} - e_{u+1}^{\boldsymbol{x}}). \tag{10}$$

Eq. (10) returns a list of new jobs under the form $a(t) = \{(0, d_i)\}_{i=1..n}$.
To compute the probability $P_a$, we introduce the following variables:

- $n$ is the number of jobs that arrive at time $t+1$;
- $k$ is the number of different job deadlines that arrive at time $t+1$;
- $n_j$ is the number of job of deadline $d_j$. It satisfies $n = n_1 + \cdots + n_k$;
- $M(\boldsymbol{x})$ is the maximal number of jobs that can arrive at a time $t+1$. We assume that our real-time system includes a *limited capacity buffer* $B$ that stores the jobs ($B$ must be smaller than $s_{\max}\Delta/C$ to guarantee feasibility). The maximal number of job arrivals depends on the previous state $\boldsymbol{x}$. This is why $M(\boldsymbol{x}) = B - n(\mathit{Shift}_u(\boldsymbol{x}, e_\alpha^{\boldsymbol{y}} - e_{u+1}^{\boldsymbol{x}}))$ is the maximal number of jobs that can arrive at time $t+1$.

We note $\mathcal{S}_{\boldsymbol{x}}$ the set of possible successors of $\boldsymbol{x}$.
The function $P_a$ satisfies different properties that are stated below. Two cases must be considered: (1) $\boldsymbol{y} \notin \mathcal{S}_{\boldsymbol{x}}$; in this case we have $P_a = 0$; (2) $\boldsymbol{y} \in \mathcal{S}_{\boldsymbol{x}}$; we distinguish several sub-cases, depending on the set of the new jobs $a(t) = \boldsymbol{y} \ominus \mathit{Shift}_u(\boldsymbol{x}, r_\alpha)$.
If $a(t) = \emptyset$, then we have:

- If $\ell_{\boldsymbol{y}} \neq \ell_{\boldsymbol{x}} + 1$, then $P_a = 0$.
- If $\ell_{\boldsymbol{y}} = \ell_{\boldsymbol{x}} + 1$, then $P_a = \mathbb{P}(a(t)) = 1 - \frac{\nu(\ell_{\boldsymbol{x}})}{\sum_{i=\ell_{\boldsymbol{x}}}^{L} \nu(i)}$.
- If $\ell_{\boldsymbol{x}} = L$ and $\ell_{\boldsymbol{y}} = 1$, then $P_a = 1$.

If $a(t) \neq \emptyset$, then we have:

- If $\ell_{\boldsymbol{y}} \neq 1$, then $P_a = 0$.
- If $\ell_{\boldsymbol{y}} = 1$, then using $C$ (the biggest job size), the general case for the probability $\mathbb{P}(a(t))$ is presented below $\forall n \in [1, M_n(x)]$.

Let us analyze the most general case: $a(t) \neq \emptyset$ and $\ell_{\boldsymbol{y}} = 1$. In a first step, we suppose that the number of jobs that arrive does not lead to a full buffer, *i.e.*, $n < M_n(\boldsymbol{x})$. We begin by analyzing the inter-arrival time values. In this situation, since there is at least one arriving job ($a(t) \neq \emptyset$), it means that we have to consider the inter-arrival time $\ell_{\boldsymbol{x}}$, which is chosen among all the possible inter-arrival times, *i.e.*, all values between $\ell_{\boldsymbol{x}}$ and $L$. This probability, which is the probability that the first job of $a(t)$ arrives, is $\frac{\theta(\ell_{\boldsymbol{x}})}{\sum_{i=\ell_{\boldsymbol{x}}}^{L} \theta(i)}$. Each additional job in $a(t)$ depends on the probability of the zero inter-arrival time, because these jobs must arrive simultaneously. For each of them the arrival probability is $\theta(0)$. Since there are $n-1$ job arrivals after the first job in $a(t)$, the probability for all these jobs is $\theta(0)^{n-1}$. Finally, since there are exactly $n$ new jobs, we multiply by the probability of non zero inter-arrival time $(1 - \theta(0))$ for the next job arrival.

Regarding the deadlines, the $a(t)$ job deadlines are independent, so the probability to have $n_i$ jobs of deadline $d_i$ is $\delta(d_i)^{n_i}$. By considering all existing deadlines, it becomes

$$\prod_{i=1}^{k} \delta(d_i)^{n_i}. \tag{11}$$

But since jobs $(0, d_i)$ with the same deadline $d_i$ are not ordered in $a(t)$, the product in Eq. (11) captures several cases that correspond to the same state. Since there are $n_i!$ possibilities for the truncated list of new jobs of deadline $d_i$, we have to multiply to the probability computation the factorial fraction, which corresponds to all possible combinations of jobs divided by all the combination for jobs of same deadline, because those are not ordered:

$$\frac{n!}{\prod_{i=1}^{k} n_i!} \tag{12}$$

These two analyses lead to the following job arrival probability for $a(t) \neq \emptyset$ and $\ell_{\boldsymbol{y}} = 1$, in the case where $n < M_n(\boldsymbol{x})$:

$$\mathbb{P}(a(t)) = \frac{\theta(\ell_{\boldsymbol{x}})\,\theta(0)^{n-1}(1-\theta(0))}{\sum_{i=\ell_{\boldsymbol{x}}}^{L}\theta(i)} \frac{n!}{\prod_{i=1}^{k} n_i!} \prod_{i=1}^{k} \delta(d_i)^{n_i} \tag{13}$$

In a second step, we consider that $n = M_n(\boldsymbol{x})$. So this situation simplifies Eq. (13) by removing the probability that there is not an extra $(|a(t)|+1)^{th}$ job:

$$\mathbb{P}(a(t)) = \frac{\theta(\ell_{\boldsymbol{x}})}{\sum_{i=\ell_{\boldsymbol{x}}}^{L}\theta(i)}\,\theta(0)^{n-1} \frac{n!}{\prod_{i=1}^{k} n_i!} \prod_{i=1}^{k} \delta(d_i)^{n_i}. \tag{14}$$

Putting everything together gives the transition probability. The fact that the transition matrix can be constructed by only using $\theta$, $\sigma$, and $\delta$ confirms the fact that $(\boldsymbol{x}, \ell_{\boldsymbol{x}})$ is a proper state of the system. This allows us to set the energy optimization problem as a Markov decision process.

# 3 Markov Decision Process

Recall that the energy consumption of the processor over one time interval $(t, t+1]$, when working at speed $s(t)$, is denoted $P_{ower}(s(t))$. The expected energy consumption from $t = 0$ up to the time horizon $T$, under any initial state $\boldsymbol{w}$, is denoted $F_0$:

$$F_0(\boldsymbol{w}) = \mathbb{E}\left(\sum_{t=0}^{T} P_{ower}(s(t))\right). \tag{15}$$

Our goal is to compute, at each time $t$, the optimal speed $s^*$ to minimize Eq. (15). This can be done by solving the following backward optimality Bellman equation [13], which becomes explicit here since the state space is finite and the transition probability matrix has been computed in Sec. 2.4. This yields the following results.

**Theorem 1** *If the state at time $t$ is $\boldsymbol{w}$, then the optimal expected energy consumption from $t$ to $T$ is:*

$$F_t^*(\boldsymbol{w}) = \min_{s\in\mathcal{A}(\boldsymbol{w})}\left(P_{ower}(s) + \sum_{\boldsymbol{w}'\in\mathcal{W}} P(\boldsymbol{w}, s, \boldsymbol{w}')F_{t+1}^*(\boldsymbol{w}')\right) \tag{16}$$

*where $\mathcal{A}(\boldsymbol{w})$ is the set of admissible speeds in state $\boldsymbol{w}$ that make sure that no job will miss a deadline at time $t$:*

$$\mathcal{A}(\boldsymbol{w}) = \left\{ s \in \mathcal{S}, s \geq \sum_{i \ s.t. \ d_i=1} (C - e_i) \right\}. \tag{17}$$

*The optimal speed $s_t^*(\boldsymbol{w})$, to be used under state $\boldsymbol{w}$ at time $t$, is any speed that achieves the* min *in Eq.* (16).

As for the infinite horizon, the long run average energy consumption over one step is $g = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\sum_{t=0}^{T} P_{ower}(s(t))$.

**Theorem 2** *The optimal average consumption $g^*$ is the solution of the fixed point Bellman equation (with bias $h^*$):*

$$g^* + h^*(\boldsymbol{w}) = \min_{s\in\mathcal{A}(\boldsymbol{w})} \left( P_{ower}(s) + \sum_{\boldsymbol{w}'\in\mathcal{W}} P(\boldsymbol{w}, s, \boldsymbol{w}') h^*(\boldsymbol{w}') \right) \tag{18}$$

*The optimal speed $s^*(\boldsymbol{w})$, to be used under state $\boldsymbol{w}$, is any speed that achieves the* min *in Eq.* (18) *(note that it does not depend on the time $t$).*

The quantities $g^*$ as well as the optimal speeds $s^*(\boldsymbol{w})$ can be computed off-line using *value* or *policy iteration* algorithms, with is quadratic complexity in the size of the state space. This can be a burden when the state space is very large. In this case, a coarser discretization of the state space can be used to reduce the size of the state space.

**Corollary 1** *Let us denote by $B$ the maximal number of jobs that can be present in the system at any time, $C$ and $\Delta$, the respective maximal job sizes and deadlines, then:*

1. *If $s_{\max} \geq BC$, then under the optimal speeds $s^*(\boldsymbol{w})$ all jobs will be completed before their deadlines.*
2. *In the particular case where jobs are released one at a time, if $s_{\max} \geq C$, then under the optimal speeds $s^*(\boldsymbol{w})$ all jobs will be completed before their deadlines.*

**Proof** *Since $B$ is the maximal number of jobs that can be pending at a time instant $t$ and $C$ is the maximal job size, we can deduce that, at each instant, the workload of the processor is at most $BC$. The worst case occurs when job deadlines are* 1. *In this situation, a constant processor speed equal to $BC$ can execute all jobs before their deadlines. Therefore, if $s_{\max} \geq BC$, there exists a speed policy that never misses a deadline. This implies that the optimal speed policy will also execute all jobs before their deadline by setting the energy cost of a deadline miss to infinity (more precisely, by setting the value of the* min *in Eq.* (16) *to infinity when the set $\mathcal{A}(\boldsymbol{w})$ is empty).*

*In the particular case where jobs are released one at a time, the maximal work quantity that can arrive at each time step is $C$. If the maximal processor speed is smaller than $C$, then we can face a job of size $C$ and deadline* 1*, so its deadline will be missed. In contrast, if the maximal processor speed is larger than $C$, then it is always possible to execute all jobs before their deadlines. Therefore, if $s_{\max} \geq C$, then the optimal speed policy $s^*(\boldsymbol{w})$ will also execute all jobs before their deadlines.* □

# 4 Comparative Analysis with Other Solutions

We focus on two main alternatives proposed in the literature to our solution ((OA) and (PACE), presented in full details in this section), which can deal with universal job features, *i.e.*, with arbitrary release times, deadlines, and sizes.

As explained in the introduction, there exist many other alternatives, but most of them are specific or ad-hoc to particular systems.

## 4.1 Optimal Available (OA) Speed Selection

The optimal available (OA) algorithm, introduced by Yao et al. [1] is an on-line speed policy that chooses the speed $s^{(\mathrm{OA})}(\boldsymbol{w}_t)$ at time $t$ as follows. In any state $\boldsymbol{w}_t$, $s^{(\mathrm{OA})}(\boldsymbol{w}_t)$ is the optimal speed in order to execute the current remaining work at time $t$, should all job sizes be equal to their WCET and should no further jobs arrive in the system. Yao et al. show that, by considering that all jobs present at time $t$ have a remaining worst possible equal to $C - e_i$ (with our notation $x_t^{(\mathrm{OA})} = (e_i, d_i)_{i=1\ldots n}$), then:

$$s^{(\mathrm{OA})} = \max_{i=1\ldots n} \frac{\sum_{j:d_j \le d_i} (C - e_j)}{d_i}. \tag{19}$$

### 4.1.1 Example showing (OA) can be sub-optimal

Consider a single job with arrival time 0, deadline 4 and size equal to 1, 2, 3 or 4 with respective probabilities $\frac{1}{4}$, $\frac{1}{4}$, $\frac{1}{4}$, and $\frac{1}{4}$.

Since (OA) only uses the worst case size to select its speed, at each time $t = 0, 1, 2, 3$, it selects the same speed: $s_t^{(\mathrm{OA})} = 1$.

If the power dissipated by the processor using speed $s$ is $P_{ower}(s) = Ks^2$ (classical for CMOS models), then the expected energy spent to complete the job under (OA) is:

$$\mathbb{E}(E^{(\mathrm{OA})}) = K\left(\frac{1}{4} + \frac{2}{4} + \frac{3}{4} + \frac{4}{4}\right) = \frac{10K}{4} = 2.5K.$$

On the other hand, in such a simple example, one can compute the speeds $s(0)$, $s(1)$, $s(2)$, and $s(3)$ at respective times $0, 1, 2, 3$ that minimize the expected energy consumption $E^*$, using the same approach as in [14]. Computing these optimal speeds boils down to a constrained convex minimization problem:

- Minimize $K(s(0)^2 + \frac{3}{4}s(1)^2 + \frac{2}{4}s(2)^2 + \frac{1}{4}s(3)^2)$ under the constraints $s(i) \ge 0$ for all $0 \le i \le 3$ and $s(0) + s(1) + s(2) + s(3) = 4$.

Using a Lagrange multiplier $\lambda$, the Karush-Kuhn-Tucker conditions are: $2s(0) = \lambda, \frac{3}{2}s(1) = \lambda, s(2) = \lambda$ and $\frac{1}{2}s(3) = \lambda$, under the constraint $s(0) + s(1) + s(2) + s(3) = 4$. The solution is $s(0) = \frac{12}{25}$, $s(1) = \frac{16}{25}$, $s(2) = \frac{24}{25}$, and $s(3) = \frac{48}{25}$, yielding the total expected energy:

$$\mathbb{E}(E^*) = \frac{1200K}{625} = 1.92K.$$

The relative over-consumption of (OA) versus the optimal energy for such a single job, $(\mathbb{E}(E^{(\mathrm{OA})}) - \mathbb{E}(E^*))/\mathbb{E}(E^*)$, is just above 30%.

Additional efficiency loss of (OA) is to be expected when the arrival times and the deadlines are taken into account and when the distribution of the size of the jobs is even more biased towards 0.

These intuitions will be confirmed by the numerical experiments displayed in Sec. 5.

## 4.2 (PACE) Speed Selection

### 4.2.1 Single Job Speed Selection

In [6], the optimal speed policy (PACE) to execute one job while minimizing the expected energy consumption is computed in closed form. The formula for the speed choice on job $J_i = (e_i, d_i)$

is as follow, with $\Theta$ the cumulative distributed function of the size of $J_i$:

$$s^{(\text{PACE})}(e_i) = \Omega(1 - \Theta(e_i))^{-1/3}, \tag{20}$$

where the normalizing constant $\Omega$ is obtained by solving the following equation that makes sure that any job of maximal size $C$ is completed before $d_i$:

$$\int_0^C \frac{1}{s^{(\text{PACE})}(w)} \mathrm{d}w = d_i. \tag{21}$$

(PACE) considers that the processor speed choices are continuous, but in practice only a finite number of speed values are available. In addition, decision times are also discrete. In the following, we will use a discrete version of (PACE) speed selection algorithm, which uses at each time instant the closest integer value to the speed computed with (PACE).

Moreover, as the size distribution is discrete here, the considered cumulative distributed function for the size is taken piece-wise affine and is constructed as follows. $\forall i \leq c \leq i+1$,

$$\Theta(w) = (F_\sigma(i+1) - F_\sigma(i))F_\sigma(i)(w - i) + F_\sigma(i), \tag{22}$$

written under the form $\Theta(w) = a_i w + b_i$ where $F_\sigma(i) = \sum_{j=0}^{i} \sigma(j)$, $a_i = (F_\sigma(i+1) - F_\sigma(i))F_\sigma(i)$ and $b_i = F_\sigma(i) - (F_\sigma(i+1) - F_\sigma(i))F_\sigma(i)i$.

From this, the normalizing constant $\Omega$ for a job with deadline $d_i$ is:

$$\Omega = \frac{3}{4d_i} \sum_{[i,i+1] \in [0,C]} \frac{1}{a_i} \left[ (1 - a_i i - b_i)^{4/3} - (1 - a_{i+1}(i+1) - b_{i+1})^{4/3} \right].$$

Therefore, the speed for a job $J_i$ with deadline $d_i$ when an amount of work $e_i$ has already been executed on $J_i$ (with $i \leq e_i < i+1$), is

$$s^{(\text{PACE})}(J_i(e_i, d_i)) = \Omega \left[1 - (a_i e_i + b_i)\right]^{-1/3}. \tag{23}$$

The speed computed by Eq. (23) is not, in general, an integer. We therefore round it to the closest available speed from the set $\mathcal{S}$:

$$round(s, \mathcal{S}) = \arg\min_{s' \in \mathcal{S}} \{|s - s'|\} \tag{24}$$

$$\tilde{s}^{(\text{PACE})}(J_i) = round(s^{(\text{PACE})}(J_i), \mathcal{S}). \tag{25}$$

### 4.2.2 Several Job Speed Selection

In the case where several jobs are present at time $t$, (PACE) executes each individual job at the speed computed for each job in isolation, as presented in the previous section. The speed of the processor is the sum of the speeds allocated to each current job. In other words, these jobs are executed in processor sharing. The resulting speed must however belong to the set $\mathcal{S}$, so the computed speed when several jobs are present at time $t$ is:

$$\min \left\{ s \in \mathcal{S} | s \geq \sum_{J_i \in \boldsymbol{x}_t} \tilde{s}^{(\text{PACE})}(J_i) \right\}. \tag{26}$$

This set could be empty, meaning that the speed $s_{\max}$ is not large enough to execute all the jobs present in the system before their deadlines. To prevent this from occurring, we add a feasibility condition: If $d_i = 1$, then $\tilde{s}^{(\text{PACE})}(J_i) = C - e_i$.

### 4.3 Sub-optimality of (PACE)

As for (OA) we exhibit an example where (PACE) does not behave very well. Since (PACE) is optimal for single jobs, as well as periodic tasks whose deadline is smaller than the period, the example will involve simultaneous job releases.

Let us consider the case where $n$ jobs $J_1, \dots J_n$ are released simultaneously, all with the same size ($c = 1$), with respective deadlines $1, 2, \cdots, n$. Since the distribution of job sizes is degenerate (all sizes are deterministic, equal to 1), the speed selected by (PACE) to execute one job with relative deadline $d$, is constant over the time interval from its release time to its deadline, equal to $1/d$. Therefore, at time 0 and for job $J_i$, the speed selected by (PACE) is $s^{(\mathrm{PACE})}(J_i) = 1/i$, so the cumulative speed at time 0 is

$$\begin{aligned} s^{(\mathrm{PACE})}(J_1) + \cdots + s^{(\mathrm{PACE})}(J_n) &= 1 + 1/2 + \cdots + 1/n \\ &= H_n \approx \log n. \end{aligned}$$

At time 1, the cumulative speed is $1/2 + \cdots + 1/n = H_n - H_1$ and so forth up to the speed used at time $n-1$, equal to $1/n = H_n - H_{n-1}$. Under the quadratic model for the power consumption ($P_{ower}(s) = Ks^2$), the total energy spent under (PACE) is

$$\begin{aligned} \mathbb{E}(E^{(\mathrm{PACE})}) &= K\left(H_n^2 + (H_n - H_1)^2 + \cdots + (H_n - H_{n-1})^2\right) \\ &= K\left(nH_n^2 + \sum_{i=1}^{n-1} H_i^2 - 2H_n\left(\sum_{i=1}^{n-1} H_i\right)\right) \\ &= K\left(nH_n\left(H_n - \frac{2(n-1)}{n}\right) + \sum_{i=1}^{n-1} H_i^2\right) \\ &= 2Kn(\log n)^2 + O(n \log n). \end{aligned}$$

Meanwhile, under the same set of jobs, (OA) will use speed 1 at each time slots $0, 1, \dots n-1$. The total energy used by (OA) to complete all jobs in that case is

$$\mathbb{E}(E^{(\mathrm{OA})}) = Kn. \tag{27}$$

When $n$ grows, the relative gain of (OA) over (PACE) grows to infinity in this case.

These two examples (Sec. 4.1.1 and 4.3) show respectively that in some cases (OA) behaves much better than (PACE), while in other cases, (PACE) is better. The numerical experiments reported in Sec. 5 confirm that optimal speeds computed by our MDP algorithm provide an energetic gain over previous solutions.

## 5 Numerical Experiments

### 5.1 General Parameters Used in the Experiments

The distributions used in the experiments are the following:

- For all job features, *i.e.*, inter-arrival times, sizes, and deadlines follow a given distribution, and are independent of each other. Their cumulative distribution functions are defined as in Sec. 4.2.
- To ensure that our state space is not to large, we consider that there exists a maximal number of jobs in the buffer of the processor at each time step (noted $M_n$ in Sec. 2.4). It will be fixed to 3 in most of our simulations.

There are $N = 1000$ simulations done for each experimental test. For each of them, we execute a job sequence over a time horizon of $T = 1000$ time steps.

## 5.2 Numerical Results

In this part we analyze the impact of the features of the jobs (*i.e.* size, inter-arrival time and deadline probability distributions) on the energy consumption of (MDP), and compare it with the energetic performance of the two other policies (PACE) and (OA) respectively. In all these experiments, we analyze the relative over-consumption. The over-consumption of (PACE) with respect to (MDP) is defined as:

$$\text{Over-consumption} = \frac{\text{Energy}_{(\text{PACE})} - \text{Energy}_{(\text{MDP})}}{\text{Energy}_{(\text{MDP})}} \tag{28}$$

The same formula is used for (OA).

### 5.2.1 Impact of Inter-arrival times

In this part we study the impact of the inter-arrival time distribution on the relative over-consumption of the two policies (OA) and (PACE) in comparison with (MDP). The experiments are done for a system with following job features:

- Job deadlines are uniformly distributed from 1 to $\Delta = 3$.
- Job sizes are uniformly distributed from 1 to $C = 4$.
- Buffer size, as defined in Sec. 2.4, is set to $M_n = 3$.

We report the over-consumption obtained by (OA) and (PACE) in Tables (1) and (2), for different inter-arrival time distributions, when the inter-arrival time $L$ is bounded by 1 (Table (1)) and when $L$ bounded by 4 with no multiple job arrivals (Table (2)). The better the knowledge of the inter-arrivals, the worse the over-consumption of (OA) vs (MDP). The trend is the same for (PACE). When there are many pending jobs, (PACE) consumes a lot of energy compared to (MDP). Indeed, (PACE) is only optimal with one job at each time: when no job arrives until the current job is completed, (PACE) is optimal as well as (MDP).

Table 1: Influence of the inter-arrival time distribution on the over-consumption of (OA) vs (MDP) and (PACE) vs (MDP) (uniform deadline and size distributions, $\Delta$=3, $C$=4, $L$=1).

| Inter-arrival time distribution | | Over-consumption | |
|---|---|---|---|
| $\tau(0)$ | $\tau(1)$ | (PACE) vs (MDP) | (OA) vs (MDP) |
| 3/4 | 1/4 | 76.56% | 3.94% |
| 1/2 | 1/2 | 65.06% | 3.42% |
| 1/4 | 3/4 | 54.71% | 5.73% |
| 0 | 1 | 44.5% | 11.12% |

Table 2: Influence of the inter-arrival time distribution on the over-consumption of (OA) vs (MDP) and (PACE) vs (MDP) (uniform deadline and size distributions, $\Delta$=3, $C$=4, 1<$L$<4).

| Inter-arrival time distribution | | | | Over-consumption | |
|---|---|---|---|---|---|
| $\tau(1)$ | $\tau(2)$ | $\tau(3)$ | $\tau(4)$ | (PACE) vs (MDP) | (OA) vs (MDP) |
| 1/4 | 3/4 | 0 | 0 | 14.7% | 3.41% |
| 0 | 1/4 | 1/2 | 1/4 | 0.37% | 2.14% |
| 0 | 0 | 1/4 | 3/4 | 0.59% | 2.18% |

### 5.2.2 Impact of Deadlines

In this part, we study the impact of the deadline distribution on the different policies. These experiments are done for a fixed inter-arrival time of 1 in Table (3) and 3 in Table (4). The size is uniform between 1 and $C = 4$ and the maximal deadline $\Delta = 3$. We test our algorithm for different deadline distributions.

Table 3: Influence of the deadline distribution on the over-consumption of (OA) vs (MDP) and (PACE) vs (MDP) (uniform size distribution, $C$=4, $L$=1).

| Deadline distribution | | | Over-consumption | |
|---|---|---|---|---|
| $d(1)$ | $d(2)$ | $d(3)$ | (PACE) vs (MDP) | (OA) vs (MDP) |
| 1/2 | 1/2 | 0 | 18.05% | 18.37% |
| 1/3 | 1/3 | 1/3 | 38.6% | 6.62% |
| 1/2 | 0 | 1/2 | 67.7 % | 6.07 % |
| 0 | 1/2 | 1/2 | 53.2% | 8.64% |
| 0 | 0 | 1 | 47.3% | 6.33% |

Table 4: Influence of the deadline distribution on the over-consumption of (OA) vs (MDP) and (PACE) vs (MDP) (uniform size distribution, $C$=4, $L$=3).

| Deadline distribution | | | Over-consumption | |
|---|---|---|---|---|
| $d(1)$ | $d(2)$ | $d(3)$ | (PACE) vs (MDP) | (OA) vs (MDP) |
| 1/2 | 1/2 | 0 | 0.36% | 0.18% |
| 1/3 | 1/3 | 1/3 | 0.55% | 2.39% |
| 1/2 | 0 | 1/2 | 0.46% | 2.39% |
| 0 | 1/2 | 1/2 | 0.11% | 7.9% |
| 0 | 0 | 1 | 26% | 52% |

One can notice in Table (3), that if deadlines get shorter, then the over-consumption of (PACE) becomes smaller. This is expected, because short deadlines imply that there are potentially less jobs present simultaneously. This is closer to ideal situation where jobs are isolated.

(OA) is not very dependent on the deadline distribution and has an identical over-consumption, except for the case when the system is heavily loaded.

For inter-arrival times all equal to 3, Table (4) shows that (PACE) is close to (MDP) in terms of energy consumption, which is due to the fact that there is only one job in the buffer at each time instant. If deadlines are large, (MDP) and (PACE) benefits for the knowledge of the probability distributions, whereas (OA), which is oblivious of the probability distributions, suffers from a significant energy over-consumption.

### 5.2.3 Impact of Job Sizes

In this part, we study the impact of the size distribution on the different policies. The experiments are done for a system with jobs of fixed deadline $d = \Delta = 3$, and same $C = 4$. To analyze the impact of size distributions, we investigate two cases where jobs form periodic tasks:

1. At each time step, there is a job that arrives in the system. The inter-arrival time is 1 with probability 1.
2. At one time step out of three, there is a job that arrives in the system. The inter-arrival time is 3 with probability 1. This condition implies that there is at most one job present in the buffer of jobs at each instant, because the inter-arrival time has the same value as the deadline.

In this part, the randomness is only on the size of incoming job, whose range is chosen as follows:

$$\mathbb{P}(C = \{1, 2, 3, 4\}) = \left\{ \frac{i_1}{10}, \frac{i_2}{10}, \frac{i_3}{10}, \frac{i_4}{10} \right\} \quad \text{with } i_1 + i_2 + i_3 + i_4 = 10 \text{ and } i_4 > 0. \tag{29}$$

We compute the over-consumption of each policies (OA) and (PACE) in comparison with (MDP) for all possible distributions satisfying Eq. (29).

**Impact of the size distribution on the consumption of** (OA) Fig. 1 represents the over-consumption of (OA) against (MDP) with $L = 1$ (left) and with $L = 3$ (right) in function of the mean value of the job sizes' distribution. In all the figures, the blue, red, and green curves depict respectively the min, average, and max values.

Whatever the inter-arrival times, (OA) converges towards (MDP) when the mean of the job sizes converges to $C$. This is because (OA) is build as if each job had an size equal to $C$.

Since (MDP) takes into consideration the size probability distribution and is an optimal policy, it is always better than (OA), as expected. The over-consumption of (OA) is smaller when $L = 1$. The reason is that, when $L = 1$, the system is more heavily loaded, requiring higher processor speeds and therefore a smaller range of available speeds for both (OA) and (MDP).

**Impact of the size distribution on consumption of** (PACE) Fig. (2) represents the over-consumption of (PACE) against (MDP) with $L = 1$ (left) and with $L = 3$ (right) as a function of the mean of the job size.

Fig (2) shows that (PACE) with an inter-arrival time of $L = 1$ is better than when $L = 3$. This observation is in line with the policy definition: Indeed, (PACE) is only optimal for a job in isolation, which is not the case in the left graph of Fig (2). When the inter-arrival time is 1, several jobs can be present in the buffer at the same time. When the inter-arrival time is 3 and the deadline is also 3, there is at most one job in the buffer at any time.

Even if (PACE) is optimal for one job in isolation, we note anyway in the right graph that (PACE) has a mean over-consumption of 20% against (MDP). These difference could be due to the fact we consider a discretized (PACE) (which is more realistic in practice, because processor speeds are finite and decision times are also discrete). When the inter-arrival time is 3, one can notice that the size distribution has an important impact on the energy consumption and this is due to the fact that the speed selected by (PACE) may be close — or not — to an

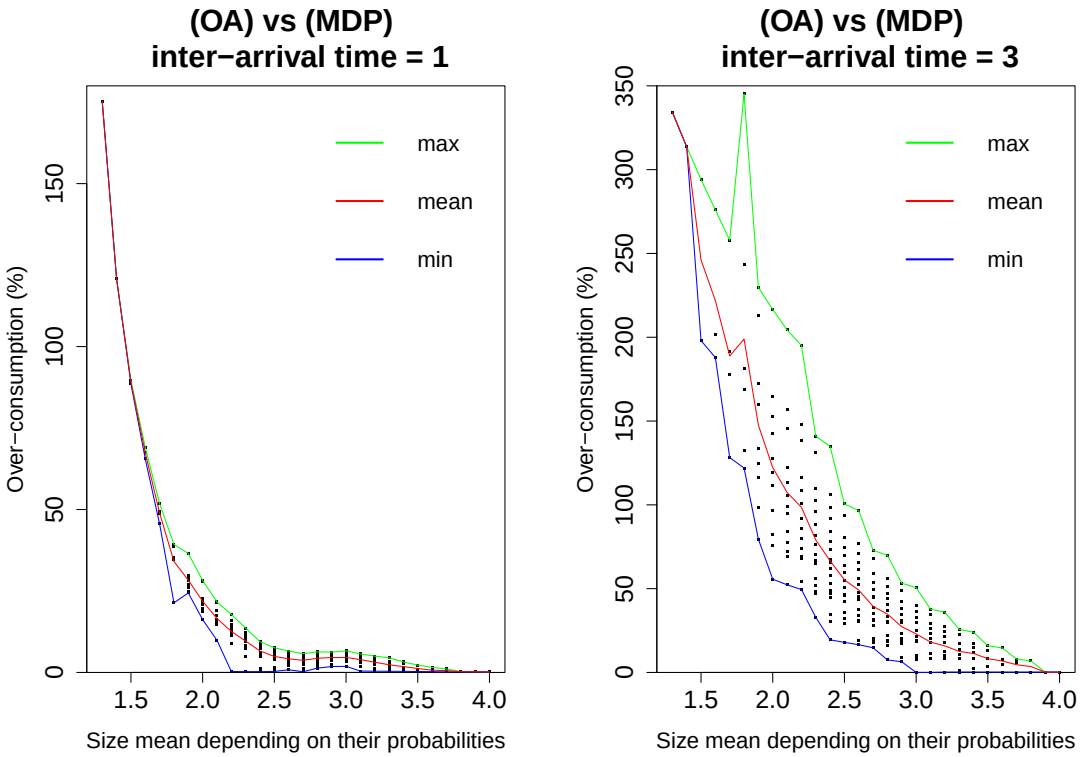



Figure 1: Influence of the size distribution on the over-consumption of (OA) versus (MDP), with fixed jobs deadline $d = 3$, fixed inter-arrival time $L = 1$ (left) or $L = 3$ (right), and a fixed buffer size $B = 3$.

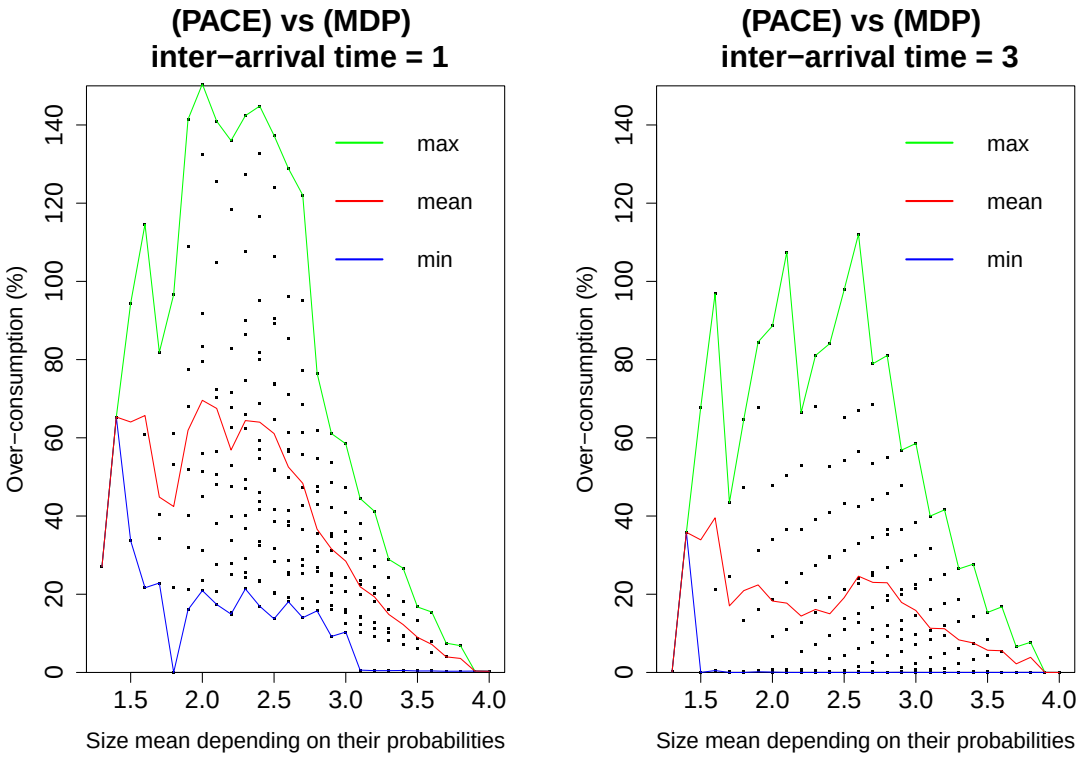



Figure 2: Influence of the size distribution on the over-consumption of (PACE) versus (MDP), with fixed jobs deadline $d = 3$, fixed inter-arrival time $L = 1$ (left) or $L = 3$ (right), and a fixed buffer size $B = 3$.

integer value. For example, with $\mathbb{P}(\{c = (1,2,3,4)\}) = \{2/10, 0, 7/10, 1/10\}$, the difference between (PACE) and (MDP) is very small, only 0.14%. In contrast, with $\mathbb{P}(\{c = (1,2,3,4)\}) = \{9/10, 0, 0, 1/10\}$, the over-consumption of (PACE) is 96.43%. This difference is mainly due to the speed discretization. Increasing the number of available processor speeds will reduce this difference.

### 5.2.4 Test with an Edge Detection Algorithm

We tested our on-line speed policy on a real life embedded system, an edge detection algorithm. It takes as input a video and produces images that represent the edge detection of one frame out of 3 from the video. This system displays a great variety for its execution time, depending both on the input data and on the initial state of the hardware. We executed it many times to build the distribution of its execution time. Fig. 3 represents the distribution of the duration for the edge detection algorithm on a video of 1 second, that produces 10 images. Since the number of different durations is important (90 different values in our example, see the top part of Fig. 3), we reduce it to 10 groups only by aggregating the values into groups as displayed in the bottom part of Fig. 3.

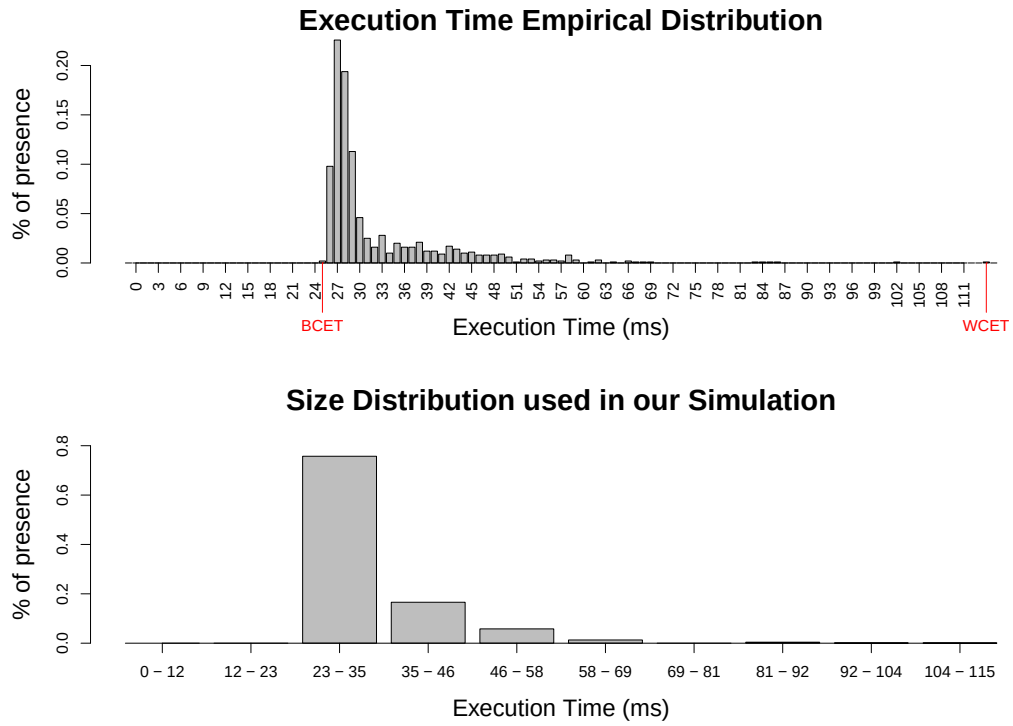



Figure 3: Distribution of execution time for the edge detection algorithm over $1,000$ executions on a $1\,s$ video. The first bar-plot depicts the distribution of the execution time, and the second is the corresponding discrete distribution used to test (MDP), (OA), and (PACE).

The over-consumption of (PACE) vs (MDP) on this example is 186%, while it is 106% for

(OA) vs (MDP). Concerning (PACE), this is mainly due to the speed discretization that is intrinsic to this algorithm. The poor performance of (OA) is expected because the mean of the size distribution ($32.28\,ms$) is significantly lower than the WCET $C$ ($114\,ms$).

For reproducibility purposes, all data and all programs used in the experimental section are available at `https://github.com/ISPASS/ISPASS.git`, and the edge detection program is available at `https://github.com/nus-mmsys/tmf.git`.

# 6 Conclusion

We have proposed a Markov Decision Process (MDP) solution to compute the optimal on-line speed policy for the *single processor hard real-time energy minimization problem*, in the *non-clairvoyant case*. The goal of this policy is to decide the speed of the processor so as to minimize the total energy consumption of the processor thanks to statistical information of the real-time jobs (release time, execution time, deadline), while guaranteeing that no job misses its deadline. Our context is more general than previous work: jobs' execution times are unknown at release time, jobs are sporadic, and several jobs can be active at the same time.

Simulations show that our (MDP) solution outperforms classical on-line solution on average, and can be very attractive when the mean value of the execution time distribution is far from the WCET, and/or when the statistical knowledge on the jobs' features is accurate.

A first future work direction amounts to reducing the time and space complexity of our algorithm, because it is exponential in the maximal deadlines of the jobs, which limits the applicability of our solution. A potential solution would be to reduce the state space by merging some "close" states of the (MDP).

A second future work direction involves using learning techniques to determine on-line the distributions of the jobs' features, and thus the processor speeds, instead of measuring these distributions off-line by profiling techniques.

# References

[1] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proceedings of lEEE Annual Foundations of Computer Science*, pp. 374–382, 1995.

[2] B. Gaujal, N. Navet, and C. Walsh, "Shortest path algorithms for real-time scheduling of FIFO tasks with minimal energy use," *ACM Trans. Embed. Comput. Syst.*, vol. 4, no. 4, 2005.

[3] N. Bansal, T. Kimbrel, and K. Pruhs, "Speed scaling to manage energy and temperature," *J. ACM*, vol. 54, no. 1, 2007.

[4] M. Li and F. Yao, "An efficient algorithm for computing optimal discrete voltage schedules," *SIAM J. Comput.*, vol. 35, pp. 658–671, 2005.

[5] J. Lorch and A. Smith, "Improving dynamic voltage scaling algorithms with PACE," pp. 50–61, 2001.

[6] J. Lorch and A. Smith, "PACE: A new approach to dynamic voltage scaling," *IEEE Trans. Computers*, vol. 53, no. 7, pp. 856–869, 2004.

[7] R. Xu, C. Xi, R. Melhem, and D. Mossé, "Practical PACE for embedded systems," in *International Conference on Embedded Software, EMSOFT'04*, (Pisa, Italy), pp. 54–63, ACM, Sept. 2004.

[8] E. Bini and C. Scordino, "Optimal two-level speed assignment for real-time systems," *Int. J. of Embedded Systems*, vol. 4, no. 2, pp. 101–111, 2009.

[9] Y. Zhang, Z. Lu, J. Lach, K. Skadron, and M. Stan, "Optimal procrastinating voltage scheduling for hard real-time systems," in *Design Automation Conference, DAC'05*, (San Diego (CA), USA), pp. 905–908, ACM, June 2005.

[10] C. Rusu, R. Melhem, and D. Mossé, "Maximizing the system value while satisfying time and energy constraints," in *Real-Time Systems Symposium, RTSS'02*, (Austin (TX), USA), pp. 246–255, IEEE, Dec. 2002.

[11] F. Gruian and K. Kuchcinski, "Uncertainty-based scheduling: Energy-efficient ordering for tasks with variable execution time," in *International Symposium on Low Power Electronics and Design, ISPLED'03*, (Seoul, Korea), pp. 465–468, ACM, Aug. 2003.

[12] R. Xu, R. Melhem, and D. Mossé, "A unified practical approach to stochastic DVS scheduling," in *International Conference on Embedded software, EMSOFT'07*, (Salzburg, Austria), pp. 37–46, ACM, Sept. 2007.

[13] M. L. Puterman, *Markov Decision Process: Discrete Stochastic Dynamic Programming*. Wiley, wiley series in probability and statistics ed., Feb. 2005.

[14] R. Xu, D. Mossé, and R. Melhem, "Minimizing expected energy in real-time embedded systems," in *International Conference on Embedded Software, EMSOFT'05*, (Jersey City (NJ), USA), pp. 251–254, ACM, Sept. 2005.

[15] B. Gaujal, A. Girault, and S. Plassart, "Dynamic speed scaling minimizing expected energy consumption for real-time tasks," Tech. Rep. HAL-01615835, Inria, 2017.

[16] P. Hilton and J. Pedersen, "Catalan numbers, their generalization, and their uses," *The Mathematical Intelligencer*, vol. 13, pp. 64–75, Mar 1991.

[17] W. Horn, "Some simple scheduling algorithms," *Naval Research Logistics*, vol. 21, no. 1, pp. 177–185, 1974.

[18] A. Marshall and I. Olkin, *Inequalitites: Theory of Majorization and Its Applications*, vol. 143 of *Mathematics in Science and Engineering*. Academic Press, 1979.

[19] D. Bertsekas and J. Tsitsiklis, *Neuro-dynamic programming*. Belmont, Mass., USA: Athena Scientific, 1996.

[20] A. Müller and D. Stoyan, *Comparison Methods for Stochastic Models and Risks*. No. ISBN: 978-0-471-49446-1 in Wiley Series in Probability and Statistics, Wiley, 2002.

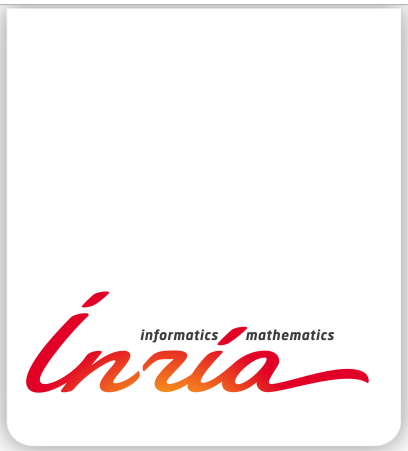
[21] H. Yun and J. Kim, "On energy-optimal voltage scheduling for fixed priority hard real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 2, no. 3, pp. 393–430, 2003.

[22] F. Gruian, "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.

[23] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *SIGOPS Oper. Syst. Rev.*, vol. 35, pp. 89–102, Oct. 2001.

[24] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, "Determining optimal processor speeds for periodic real-time tasks with different power characteristics," in *Euromicro Conference on Real-Time Systems, ECRTS'01*, (Delft, The Netherlands), pp. 225–232, IEEE Computer Society, June 2001.

[25] J. Mao, C. Cassandras, and Q. Zhao, "Optimal dynamic voltage scaling in energy-limited nonpreemptive systems with real-time constraints," *IEEE Trans. Mob. Comput.*, vol. 6, no. 6, pp. 678–688, 2007.




**RESEARCH CENTRE**
**GRENOBLE – RHÔNE-ALPES**

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399