Online optimization in dynamic real-time systems

Stéphan Plassart

Présentée par

Stéphan PLASSART

Thèse dirigée par **Bruno GAUJAL**
et par **Alain GIRAULT**

préparée au sein du **Laboratoire d'Informatique de Grenoble**
et de l'École Doctorale **MSTII**

Online energy optimisation for real-time systems.

Optimisation d'énergie en-ligne pour des systèmes temps-réel

Thèse soutenue publiquement le **16 juin 2020**,
devant le jury composé de :

Abstract

The energy consumption is a crucial issue for real-time systems, that's why optimizing it online, *i.e.* while the processor is running, has become essential and will be the goal of this thesis. This optimization is done by adapting the processor speed during the job execution. This thesis addresses several situations with different knowledge on past, active and future job characteristics. Firstly, we consider that all job characteristics are known (the offline case), and we propose a linear time algorithm to determine the speed schedule to execute n jobs on a single processor. Secondly, using Markov decision processes, we solve the case where past and active job characteristics are entirely known, and for future jobs only the probability distribution of the jobs characteristics (arrival times, execution times and deadlines) are known. Thirdly we study a more general case: the execution is only discovered when the job is completed. In addition we also consider the case where we have no statistical knowledge on jobs, so we have to use learning methods to determine the optimal processor speeds online. Finally, we propose a feasibility analysis (the processor ability to execute all jobs before its deadline when it works always at maximal speed) of several classical online policies, and we show that our dynamic programming algorithm is also the best in terms of feasibility.

Résumé

La consommation d'énergie est un enjeu crucial pour les systèmes temps réel, c'est pourquoi l'optimisation en ligne, c'est-à-dire pendant l'exécution du processeur, est devenue essentielle et sera le but de cette thèse. Cette optimisation se fait en adaptant la vitesse du processeur lors de l'exécution des tâches. Cette thèse aborde plusieurs situations avec des connaissances différentes sur les caractéristiques des tâches passées, actuelles et futures. Tout d'abord, nous considérons que toutes les caractéristiques des tâches sont connues (le cas hors ligne), et nous proposons un algorithme linéaire en temps pour déterminer les choix de vitesses pour exécuter n tâches sur un seul processeur. Deuxièmement, en utilisant les processus de décision de Markov, nous résolvons le cas où les caractéristiques des tâches passées et actuelles sont entièrement connues, et pour les futures tâches, seule la distribution de probabilité des caractéristiques des tâches (heures d'arrivée, temps d'exécution et délais) est connue. Troisièmement, nous étudions un cas plus général : le temps d'exécution n'est découvert que lorsque la tâche est terminée. En outre, nous considérons également le cas où nous n'avons aucune connaissance statistique des tâches, nous devons donc utiliser des méthodes d'apprentissage pour déterminer les vitesses optimales du processeur en ligne. Enfin, nous proposons une analyse de faisabilité (la capacité du processeur à exécuter toutes les tâches avant leurs échéances quand il fonctionne toujours à vitesse maximale) de plusieurs politiques en ligne classiques, et nous montrons que notre algorithme de programmation dynamique est également le meilleur en terme de faisabilité.

Remerciements (Acknowledgments)

Je souhaite tout d'abord remercier mes codirecteurs de thèse Bruno Gaujal et Alain Girault dont la disponibilité, la patience et les conseils ont été essentiels à la réussite de ce travail. Leur complémentarité a fait merveille tant sur le plan technique qu'organisationnel : je leur en suis très reconnaissant.

Un grand merci à tous les membres du Jury de thèse qui, malgré les reports et conditions difficiles de soutenance en lien avec la pandémie de Covid19, ont pu apporter un retour et des commentaires très pertinents sur l'ensemble de ma thèse. Merci à Florence Maraninchi d'avoir accepté la Présidence de ce jury et aux rapporteurs Liliana Cucu-Grosjean et Jean-Philippe Gayon : leur travail a été compliqué compte tenu de cette période de confinement.

J'ai également une pensée toute particulière pour les membres des équipes au sein desquelles j'ai été hébergé (parfois même au sens propre !), que ce soit l'équipe Polaris/Datamove au bâtiment Imag sur le campus (que certains appellent datapol ou polamove sans rentrer ici dans ce débat sémantique), ou que ce soit dans l'équipe Spades à l'Inria de Montbonnot.

Remerciements aussi aux anciens membres partis vers d'autres horizons, avec qui j'ai souvent eu l'occasion d'échanger. Les conditions dans lesquelles j'ai pu travailler ont été optimales, et j'aimerais mettre en avant la qualité de la recherche au sein du laboratoire (LIG) et de l'Inria, sans oublier le LabEx Persyval-Lab.

Enfin une pensée pour mes proches qui m'ont toujours soutenu, avec une émotion particulière à l'endroit de ma grand mère (qui vient de nous quitter) et de ma soeur handicapée.

Contents

1 Introduction

1.1 Context

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. In particular, we focus on hard real-time system (HRTS), that consists of a generally infinite sequence of independent jobs that must be executed onto some hardware platform before some strict deadline. Such systems are found everywhere today: in energy production, in transport (automotive, avionics, ...), in embedded systems, to name only a few application domains.

Among numerous hardware and software techniques used to reduce energy consumption of a processor, supply voltage reduction, and hence reduction of CPU speed, is particularly effective. This is because the energy consumption of the processor is a function at least quadratic in the speed of the processor in most models of CMOS circuits. Nowadays, variable voltage processors are readily available and a lot of research has been conducted in the field of Dynamic Voltage and Frequency Scaling (DVFS). Under real-time constraints, the extent to which the system can reduce the CPU frequency (or speed in the following) depends on the jobs' features (execution time, arrival date, deadline) and on the underlying scheduling policy. Several algorithms have been proposed in the literature to adapt processor speed dynamically by using DVFS technique.

1.2 Problematic

We address in this thesis the *single processor hard real-time energy minimization problem*. It consists in choosing, for each real-time job released in the system, a processor speed to execute this job, 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 completion. Nevertheless, several approaches have been proposed to tackle the hard real-time energy minimization problem under several assumptions on the processor and the jobs to be executed.

In this thesis, different solutions are proposed to reduce the energy consumption. Each of them is classified according to the context, in which it is placed.

One context is the *offline* case, where we know all jobs and their features. It means that at each instant t, one knows past jobs (all completed jobs at t), active jobs (all jobs released before or at t and not finished), and also jobs that arrive in the future (all jobs of release time strictly greater than t). It is presented in Chapter 3, and solves the classical problem of minimizing offline the total energy consumption required to execute a set of n real-time jobs on a single processor with varying speed. The goal is to find a sequence of processor speeds, chosen from a finite set of available speeds, so that no job misses its deadline and the energy consumption is minimal. Such a sequence is called an optimal *speed schedule*. In Chapter 3, we present the first *linear time* algorithm that solves this problem. The time complexity of our algorithm is in $\mathcal{O}(n)$, where n is the number of jobs, to be compared with $\mathcal{O}(n \log(n))$ for the best known solutions from the literature. Besides the complexity gain, the main interest of our algorithm is that it is based on a completely different idea: instead of computing the *critical intervals*, it sweeps the set of jobs and uses a *dynamic programming* approach to compute an optimal speed schedule. Our linear time algorithm is still valid (with some changes) with an arbitrary power function (not necessarily convex) and with arbitrary switching times.

Another context is the *online* case, which represents the core of this thesis (Chapters 4 to 7). In this situation, we discover progressively, when time goes by, the new jobs that arrive. In other terms, at each time t, the past jobs and their features are known. However, active jobs or future jobs are only *partially known*. The knowledge available on the active jobs depends on the situation we focus on, while the knowledge available on the future jobs is a *statistical*. In practice, such statistical knowledge can be obtained by observing and profiling the system.

1.3 Summary of the Contributions

Different information structures are used in this thesis, which represent practically many applicative situations in HRTSs. The differences between these structures lie in the information knowledge we consider for active jobs and future jobs, knowing always all feature about past jobs.

One situation is the case where the HRTS being considered is composed of several periodic tasks, but where each task has 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. This is frequent in embedded systems.

Another situation is the case, where periodic tasks have an unknown jitter. By knowing a probabilistic distribution on the jitter values, the energy consumption can be improved by determining more quickly all the jitters of each task.

A last situation is the sporadic tasks case, *i.e.* the case where tasks can arrive at any time. In this case, the job features are observed over a certain period to estimate the statistical properties of the jobs. This happens, for example, when jobs are dependent on external parameters. For example, a job in can be triggered by the external environment (*e.g.* a pedestrian crossing in the case of an autonomous car).

In this thesis, we will study three different online cases. All the proposed solutions are based on Markov Decision Processes (MDP). One of the difficult aspects of this thesis is to precisely define the MDP, along with its state space. Indeed, an HRTS is subject to strict timing constraints on jobs execution: no job can miss its deadline! Therefore this requires having constraints on our MDP. The strategy we use in this thesis is to incorporate these constraints in the structure of the state of the MDP. We have also to keep a close look on the state space size of the MDP, which is a crucial issue because if can become very large.

The first two online cases, solved with a MDP, require statistical knowledge of the system. Chapter 4 studies the *Clairvoyant* case, *i.e.* the case where the execution times of jobs and their deadlines are known when the jobs are released. In this chapter, the features of active jobs (deadlines and execution times) are entirely known, and one has a statistical knowledge on future jobs. With this knowledge, we compute the optimal online speed scaling policy to minimize the energy consumption of a single processor executing a finite or infinite set of jobs with real-time constraints. Several qualitative properties of the optimal policy are proved: monotonicity with respect to the jobs parameters, comparison with online deterministic algorithms. Numerical experiments in several scenarios show that our solution performs well when compared with offline optimal solutions, and out-performs online solutions oblivious to statistical information on the jobs.

Then Chapter 5 tackles a more general case, the *Non-clairvoyant* case, *i.e.*, the case where the actual execution time of the jobs is unknown when they are released. The knowledge of active jobs consists in the job deadlines and the distribution of the job execution times, therefore we have less information than Chapter 4. As a consequence, the *actual* execution time of jobs are only discovered when they finish. Regarding future jobs, the known information is identical as in Chapter 4: a statistical knowledge. 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 solution solves this problem optimally, in the discrete case. Compared with approaches from the literature, the gain offered by our speed policy ranges from a few percent when the variability of job characteristics is small, to more than 100% when the job characteristics distributions are far from their worst case.

The third case presented in Chapter 6 and 7, is the case where we have the less information available about future jobs. Indeed, in these two chapters we assume that there is no knowledge on jobs: thanks to learning technique, we compute the optimal speed schedule that executes the jobs while minimizing the energy consumption. Regarding active jobs, we assume in these two chapters that we are in the clairvoyant case: all features (execution time and deadline) are known at release time. In Chapter 6, we present how to learn one key parameter of the MDP – the transition probability matrix – and then we can determine by a value iteration algorithm the optimal speed schedule. Then, Chapter 7 investigates how to learn directly the average optimal energy consumption. Once this energy is determined, we can deduce the optimal speed schedule. These two chapters show that the convergence towards the desired transition probability matrix (resp. towards the optimal energy consumption) is very long to reach.

Still, our simulations show that, in practice, the optimal speed policy is reached reasonably quickly, and much faster in the case where the learning is done for the transition probability matrix (see Chapter 6).

The last chapter addresses the of *feasibility* of online speed policies. Feasibility is the ability for a processor to execute any sequence of jobs while meeting the two main constraints: the processor speed is always below its maximal speed and no job misses its deadline. In this chapter, we analyze the feasibility of different speed policies for single-processor HRTSs, both the one presented in this thesis (in Chapter 4), based on dynamic programming, but also those from the literature. We compute the feasibility region of four existing online speed policies in function of the maximum job size and of the maximum relative deadline. We do so for the following online speed policies: Optimal Available (OA) [YDS95], Average Rate (AVR) [YDS95], (BKP) [BKP07], and the Markovian Policy based on dynamic programming (MP) from Chapter 4. Our theoretical results show that (MP) achieves the better feasibility. This reinforces the interest of (MP) our policy that is not only optimal for energy consumption (on average) but is also optimal regarding feasibility.

1.4 Thesis Structure

This manuscript is composed of 8 different chapters, that are organized as follows. Chapter 2 introduces the general model used during all the manuscript. In each subsequent chapter, the precise model will be recalled to help the reader. Chapter 3 details our offline algorithm to minimize the energy consumption by adapting the speed processor. The four following chapters (4 to 7) address the online case. As the chapters progress, the information on future jobs decreases.

- **Chapter 4**: Active jobs are entirely known at t (deadlines and execution times) while only some probability knowledge on job features is known for future jobs.
- **Chapter 5**: At time t, the deadline of active jobs are known, while only a probability distribution of their execution time is known. As in Chapter 4, future jobs are known only statistically.
- **Chapter 6**: For active jobs, deadlines and execution times are known (as in the clairvoyant case). For future jobs, there are no information and the goal is to discover the jobs and their features by learning the transition probability matrix.
- **Chapter 7**: For active jobs, it is also the clairvoyant case. No informations are available for future jobs, and the goal is to learn the energy cost.

Our final contribution is reported in Chapter 8, where we study the feasibility of our method, based on dynamic programming, by comparing it with the feasibility of existing methods in the literature.

1.5 Reading Order

Each chapter is based on a different paper, therefore the state of the art can be recalled at the beginning of some chapters.

Some chapters are independent from the others. For this reason we make explicit in Fig 1.1 the "reading dependency order". It is a partial order, meaning that this manuscript does not necessarily need to be read in a linear way.

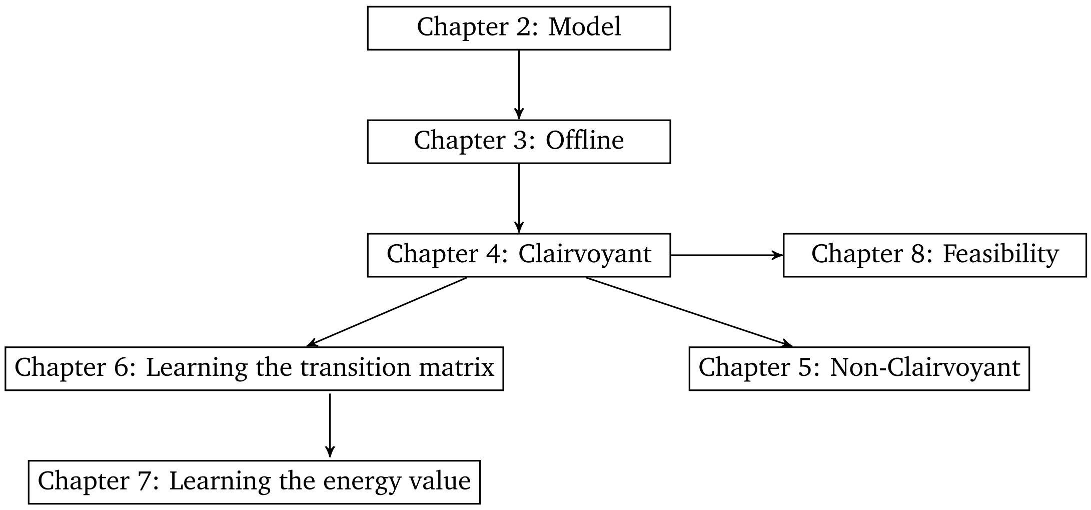

Chapter 2: Model
Chapter 3: Offline
Chapter 4: Clairvoyant
Chapter 8: Feasibility
Chapter 6: Learning the transition matrix
Chapter 5: Non-Clairvoyant
Chapter 7: Learning the energy value

Figure 1.1.: Manuscript structure and reading order: Each black arrow is a reading dependency.

All chapters require reading Chapter 2 that focuses on model presentation and problem statement. Then reading Chapter 3 gives to the reader some intuitions on the model we use in Chapter 4, because the models are very close and they have the same goal: minimizing the energy consumption by adapting the processor speed during the time. The difference is that, in Chapter 3, we consider that all jobs are known at any time, whereas in Chapter 4 we have a statistical knowledge with *clairvoyant* jobs. The algorithm presented in Chapter 4 is compared to those one of the literature in term of feasibility in Chapter 8. Chapter 8 requires the reading of Chapter 4, hence the black arrow between the two. Then, two other studies are done and require the reading of Chapter 4: one that analyses the energy minimization with statistical knowledge with *non-clairvoyant* jobs (Chapter 5), and another that corresponds to Chapter 6 and 7, where nothing is known about jobs. It is better to read Chapter 6 before Chapter 7 because it presents some definitions and structures that are used in Chapter 7.

Recall that, as we "travel down" along this graph, the amount of knowledge available on the active jobs decreases, from a complete knowledge in Chapter 3 to no knowledge at all in Chapter 7.

2 Model Presentation and Problem Statement

In this chapter, all the parameters that are needed to define a real-time system are presented. These parameters consist of job characteristics (arrival time, size and deadline) and processor characteristics (for example speed and power). We then present the main issue of this thesis: the energy minimization problem.

We consider a *hard real-time system* (HRTS) where one uni-core processor that executes a set $\mathcal{S}$ of real-time tasks. An HRTS is a system where the functionality ("what" the system does) is as important as the timing ("when" the system does something). The reason is that HRTSs operate in safety critical context where a failure can cause huge damage, including human lives (*e.g.* in avionics). It follows that tasks' deadlines can *never* be missed.

In our case, an HRTS is defined by:

- A set of *tasks*, where each task is composed of *jobs*. Tasks are either periodic (task composed of jobs that arrive periodically), or sporadic (task composed of jobs that arrive at any time). In this thesis, we will not consider the tasks but directly the set of all jobs submitted to the processor.
- A uni-core processor.
- A *scheduling policy*, that fixes for a time instant the job to execute and the processor speed at which the processor runs.

The following section will present in detail each parameters and resulting properties.

2.1 Job Description

A job J_i, $i \in \mathbb{N}$, is characterized by the triplet (τ_i, c_i, d_i), where:

- τ_i is the *inter-arrival time* of job J_i, *i.e.* the time that elapsed between the arrival of job J_{i-1} and J_i, with $\tau_1 = 1$ by convention.
- c_i is the *Worst Case Execution Time* (WCET) of job J_i when the processor is running at speed 1. This definition of the WCET corresponds to the number of operations to execute J_i on the processor. Strictly speaking, it will be not considered as a time in the following but rather as a *work quantity* to be executed, which we will also call a *size*.
- d_i is the *relative deadline* of job J_i, *i.e.* the amount of time given to the processor to execute J_i.

From the τ_i values, we can reconstruct the release time r_i of each job J_i as:

$$r_i = \sum_{k=1}^{i} \tau_k \qquad \forall i \geq 1 \tag{2.1}$$

During all this thesis, we will only use the notation of release times of jobs for the sake of simplicity, therefore the job set $\mathcal{J}$ can be rewritten as follows:

$$\mathcal{J} = \{J_i(r_i, c_i, d_i), i \in \mathbb{N}\}$$

In some chapters, the absolute deadline $D_i = r_i + d_i$ will also be used.

Throughout the document, we will study different cases, which differ on the amount of information on the jobs' characteristics that is available at design time, ranging from everything (Chapter 3) to nothing (Chapters 7 and 6).

2.2 Processor Description

2.2.1 Processor Features

The processor considered is a uni-core mono-processor, which is equipped with several integer *speeds*. There are at least two speeds: the speed 0, that corresponds to the idle state of the processor, and the maximal speed of the processor $s_{\max} \in \mathbb{N}$. $\mathcal{S} \subset \mathbb{N}$ is the finite set of the available processor speeds. Switching from one speed to another usually implies some cost in time and in energy. We will come back to these costs at the end of this section.

The processor allows *preemption*. Preemption means that the scheduler may interrupt a job in progress at any time to allow another job to execute. Such decision is usually based on the *priority* of the jobs, with lower priority jobs being preempted by higher priority ones. Priorities can be assigned either *statically* or *dynamically*. In our context, we use the dynamic scheduling policy Earliest Deadline First (EDF) [LL73], which orders the active jobs by decreasing relative deadline, and assigns the highest priority to the job with the shortest relative deadline, *i.e.* the earliest deadline.

Preemption implies some switching context cost, which must be taken into consideration. For the sake of simplicity, we will consider that both the context switching cost and the speed switching costs are null. However, our results also hold when these costs are non null, and in each relevant chapter we will present how to enhance the model to take them into account.

2.2.2 Power Description

In these hard real time systems (HRTS), the central processing unit (CPU) power consumption is crucial. It is composed of three parts: the dynamic power consumption, the static power consumption, which is composed of the short-circuit power consumption, and the power loss due to transistor leakage:

$$P_{CPU} = P_{Dyn} + P_{Sc} + P_{Leak}$$

The first power P_{Dyn} depends on the activity of logic gates in a CPU, and is due to the fact that transistors change their states. This power is nearly proportional to the CPU frequency and as a consequence to the processor speed, and is:

$$P_{Dyn} = \alpha C V_{dd}^2 s, \tag{2.2}$$

where C is the load capacitance, V_{dd} is the average voltage, and s is the processor speed. When we decrease the frequency of a circuit, we can also decrease its supply voltage V_{dd}. By Eq (2.2), the dynamic part depends directly linearly on the frequency, but also squarely on the supply voltage V_{dd}, which also depends on the frequency.

This is the purpose of Dynamic Voltage and Frequency Scaling (DVFS), a technique used in most modern processors. DVFS allows both the frequency of the processor and its supply voltage to be decreased. In this case, the dynamic power follows is such that $P_{dyn} \approx \alpha' s^\beta$, with $\alpha', \beta \in \mathbb{R}$ and $2 \leq \beta \leq 3$.

In other words, among the numerous hardware and software techniques used to reduce energy consumption of a processor, DVFS is particularly effective. As we explained before, the energy consumption of the processor is a function at least quadratic in the speed of the processor.

Throughout this thesis, we will use the function $P_{\mathit{ower}}(\cdot)$, which is defined as follows:

$$P_{\mathit{ower}} : s \to P_{\mathit{ower}}(s).$$

This function represents the power consumption of the processor when it runs at a speed s. No assumption is made on the power function, however in some chapters, for example in Chapter 4, we will study the impact of the P_{ower} function characteristics on our problem. In particular, we will look at the impact of a convexity hypothesis of the power function on the considered model.

The question that arises from this power function definition is how to take advantage of DVFS to decrease the energy consumption in HRTSs. The general idea is to adapt dynamical the processor speed during the execution of the system.

2.3 Policy Description

2.3.1 Policy Definition

The notion of *policy* is a central notion in this document, it determines the job executed at any time t and also at which speed it will be executed.

Definition 2.1 (policy). *A* policy *π is a function that assigns, at time t with the history $\mathcal{H}_t$, a speed s to the processor, and also determines the job J that is executed at time t:*

$$\pi(\mathcal{H}_t, t) = \{s, J\}\,. \tag{2.3}$$

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

$$\mathcal{H}(t) = \{(\tau_i, c_i, d_i)\,|\,r_i \leq t\} \cup \{s(u), J_u | u \leq t\}. \tag{2.4}$$

2.3.2 Feasibility and Schedulability

At any time $t \in \mathbb{R}$, several jobs may be active (*i.e.* released and not yet finished). In this case we must choose which job to execute first on the single-core processor. This ordering is known as a *schedule* and the policy for making this choice is known as the *scheduling policy*.

Let us define the notion of *feasibility*, which is an important issue in HRTS. Informally, a policy is feasible if it can execute all jobs before their respective deadline.

Definition 2.2 (policy's feasibility). *A* policy *π is feasible for a set of jobs $J = \{(r_i, c_i, d_i)\}_{i \in \mathbb{N}}$ if, when the processor uses the policy $\pi(t)$ at each time t, then each job (r_i, c_i, d_i) is executed before its deadline:*

$$\pi \text{ is feasible} \iff \left(\sup_{t \in \mathcal{T}} \pi(t) \leq s_{\max} \right) \wedge \text{ no missed deadline.} \tag{2.5}$$

Definition 2.3 (schedulability). *A set of job $J = \{(r_i, c_i, d_i)\}$ is schedulable if there exists a policy π that is feasible over J.*

2.3.3 Speed Changes

Two different notions have to be defined:

- The notion of speed *decision* instant: It is the time instant when the policy decides the speed that the processor will use in the future.
- The notion of speed *changing* instant: It is the moment when the processor can change its speed.

The number of speed changing instants is always larger than the number of speed decision instants, because it is possible that at one speed decision instant, the speed policy decides several speed changing instants.

For this document, we consider that speed decision instants and speed changing instants are integers. Throughout the thesis, we will study the relevance of this assumption and will discuss of the impact, or not, on the solution of our problem.

In particular, in Theorem 4.1 of Appendix 4.7 of Chapter 4, we prove that if the processor speeds are consecutive and integer, if the power function is convex, and if the job features are integers, then restraining to integer speed changing instant does not lead to a loss of optimality.

2.4 Problem Statement

Different cases are studied in this thesis, which depend on the information known about the jobs during the evolution of the HRTS. As presented in Chapter 1, we classify the type of available information at any time t according to the knowledge we have on three subsets of jobs: past jobs (*i.e.* jobs completed at t), active jobs (*i.e.* jobs released at of before t and not yet completed), and future jobs (*i.e.* jobs that will be released after t). All the cases presented from Chapter 4 to Chapter 7 assume that, at time t, all the past jobs and their features are known. The difference will lie in the information of active and future jobs.

Two different situations are considered:

- The *offline* case: The system is entirely known, *i.e.* all the characteristics (release times, deadlines, and sizes) for all the jobs are known. In this particular case, the number of jobs is *finite*, and all the history $\mathcal{H}(t)$, $\forall t \in \mathbb{N}$ is known at the outset. Each job is also assumed to take exactly its size to complete. In this context, the processor speeds are chosen *before* the system execution, that is, offline.

- The *online* case: At time t, only the release times of past and active jobs are known. Here the number of jobs can be finite or infinite. Three cases are studied, which make more or less strong assumptions about active and future job features.

  - The *clairvoyant* case: The active jobs are entirely known, meaning that $\mathcal{H}(t)$ in Eq. 2.4 is known at time t. In contrast, the future jobs are only known *statistically*; it means that we assume that the triplets $(r_i, c_i, d_i)_{i\in\mathcal{J}}$ are *random variables*, defined on a common probability space, whose joint distribution is known (for example by using past observations of the system): $\mathbb{P}(r_i = t, c_i = c, d_i = d)$ is supposed to be known for all t, c, d.

  - The *non-clairvoyant* case: Only the deadlines of the active jobs are known, hence $\mathcal{H}(t)$ in Eq. 2.4 is partially known at time t: for each active job J_i, the deadline d_i is known, the maximal work quantity c_i is known, but the actual work quantity required by the job will only be "discovered" when the job completes (of course, it is less than c_i). Also, and the portion J_i that already has been executed. Concerning the actual execution time, only its distribution is known. Finally, the future jobs are known as in the clairvoyant case, *i.e.* statistically.

- The *learning* case: The past and active jobs are known as in the clairvoyant case. Eq. 2.4 is satisfied with c_i and d_i known. However for future jobs, we have no information at all.

Regarding the processor speeds, ship manufacturers actually propose processors that have only a finite number of frequency, and so a finite number of speed. This is why we will consider in most chapters that processor speeds are discrete and bounded by a maximal speed $s_{\max}$. This speed model choice is due to physics reality on processors.

In this thesis, we present not only the case where the number of jobs is infinite but also the case where we have a finite set of jobs. This distinction between the finite and infinite cases is useful because it will lead to different algorithms.

The energy consumption depends on the power dissipated by the processor. As we explained before, we focus on the dynamic energy consumed by the processor, denoted E, also called the *cost function*. In the finite case, where T is the *time horizon* of the system execution, it is defined as:

$$E = \int_0^T P_{ower}(s(t))\mathrm{d}t \tag{2.6}$$

In the infinite case, the average energy is defined as follows:

$$E = \lim_{T \longrightarrow \infty} \frac{1}{T} \int_0^T P_{ower}(s(t))\mathrm{d}t \tag{2.7}$$

Since our goal in this thesis is to minimize the energy consumption of a processor with varying speed, while satisfying the condition that jobs are executed before their deadlines, then two questions arise:

- Question 1: What is the job scheduling policy to choose to be sure that all jobs are executed before their deadline?
- Question 2: What is the optimal speed choice at each instant?

Question 1 will be answered directly in this chapter, in Section 2.4.1, and Question 2 will be the central subject of this thesis.

2.4.1 EDF Optimality

When the processor speed remains constant, the *Earliest Deadline First* (EDF) preemptive scheduling policy is *optimal*, meaning that if a set of jobs is schedulable with any policy π, then it is also schedulable by EDF. Recall that a set of jobs is schedulable by a policy π if and only if, when the processor executes π, all the jobs complete before their respective deadlines (see Def. 2.3). To the best of our knowledge, the optimality of (EDF) has been proved by [Hor74] in the case where the processor speed remains constant.

It follows that (EDF) could be a good candidate to solve Question 2. However in this thesis, the processor can change at each time instant, and, to the best of our knowledge, there is no similar proof of optimality of (EDF) in this case. This is why we propose in Appendix A.1 a proof of the optimality of (EDF) with a varying speed processor. Our proof is based on the proof done by Horn [Hor74] and generalizes it.

Proposition 2.1. *When the processor speed can change at each time instant, (EDF) minimizes the maximum lateness of any set of jobs, as described in Section 2.1.*

Optimal schedulability is equivalent to having a maximum of lateness equal to 0, hence Property 2.1 is equivalent to stating that (EDF) is optimal.

From now, we will choose the scheduling policy (EDF), therefore Question 1 from Section 2.4 is solved. It remains to determine the speed used by the processor at each instant (Question 2). As a consequence, the function $\pi(\mathcal{H}_t, t)$ can be simplified: it will assign only the speed s to use at time t under history $\mathcal{H}_t$, *i.e.* $\pi(\mathcal{H}_t, t) = s$.

In the document, we will often use $\pi(t)$ to simplify the notation, but one should keep in mind the fact that, in full generality, the speed selected at time t may depend on t, the jobs that arrived before t, and the speeds selected before t.

Since the maximal speed of the processor is $s_{\max}$, with this simplified notation, any speed policy π must satisfy the following constraint:

$$\forall t, \forall J,\ 0 \leq \pi(\mathcal{H}_t, t) \leq s_{\max}. \tag{2.8}$$

2.4.2 Speed Policy

We have claimed in Section 2.4.1 that (EDF) is optimal, therefore now we can only focus on the speed policy problem, which is defined as follows:

> *Find online speeds $s(t)$ (*i.e., *$s(t)$ can only depend on the history $\mathcal{H}(t)$) in order to minimize the cost function under the constraint that no job misses its deadline.*

Jobs arrive at integer time steps t, because all the job features are integers. At first, we will investigate the case where the processor can change speed only at integer time steps. However, we will prove in Appendix 4.7 of Chapter 4 that this restriction does not impact optimality, because there always exists an optimal speed schedule where speed changes occur at integer time steps.

As said in Section 2.4, the system history $\mathcal{H}(t)$ depends on the available information on active jobs. In Chapter 3, it consists of all the jobs and their characteristics: release time, deadline and execution time. Past, active, and future job features are entirely known. In Chapter 4, $\mathcal{H}(t)$ consists of all jobs released before t with deadline and execution time data. Past and active job features are known. Future job characteristics are only known statistically. In Chapter 5, $\mathcal{H}(t)$ consists of all jobs released before t with their deadline value, the distribution of the execution time and also the job part that has been already executed for jobs not finished at time t. Past job features are known. For active job, we know the deadline, but only the distribution of the execution time. In addition, the work already executed on job is known. Future job features are also known statistically. In Chapter 6 and 7, $\mathcal{H}(t)$ consists only of the jobs that are finished. In these two chapters, past job features are known, but no statistical information about future and active job features are provided. For active job, only the deadline and the work already executed on these jobs are known.

2.5 Notations Summary

J_i	Job number i
$r_i, c_i, d_i \in \mathbb{N}$	Release time, size, and relative deadline of job i
D_i	Absolute deadline of job i
Δ	Bound on all relative deadlines
C	Bound on the work amount arriving at any time t
$\mathcal{J}_{C,\Delta}$	Set of all sequences of jobs with bounds C and Δ
$\mathcal{S}$	Set of processor speeds
$s_{\max}$	Maximal speed of the processor
$\mathcal{A}(\boldsymbol{x})$	Set of possible speeds in state $\boldsymbol{x}$
T	Time horizon
$\pi(t), \pi_t$	Speed used by the processor at time t under policy π
w_t^π	Remaining work under speed policy π at time t
$\mathcal{W}$	State space of remaining work
ℓ_i	Remaining amount of work to complete J_i at time t
(OA)	Optimal Available policy
(MDP)	Markov Decision Process
$\boldsymbol{x}$	State of the (MDP)
$\mathcal{X}$	State space of the (MDP)
$A(t)$, a	Work arriving at t
$P_{ower}(\cdot)$	Power function
γ	Discount factor
E	Energy consumption of the processor
$s^{(\mathrm{OA})}(w)$	speed chosen under (OA) policy for the state w

Table 2.1.: Notations used throughout the thesis (part 1).

$\hat{P}_n$	Estimated transition probability matrix by (PL) after n learning steps
P	Real transition probability matrix
$\hat{v}_n$	Average estimated energy cost after n samples of (PL)
v^*	Optimal average energy cost
$\hat{V}_n^\gamma$	Discounted estimated energy cost after n samples of (PL)
$V^{*,\gamma}$	Optimal discounted energy cost
R_{max}	Energy cost of the maximal speed
μ_t	Proportion of time spend in state w under speed policy s_t
μ^*	The stationary policy under the optimal policy for the exact distribution
$\phi_{\mathcal{N}(0,1)}(\cdot)$	Cumulative distribution function of the standard normal distribution
σ	Standard deviation
$q^*(w,s)$	Minimal discounted energy consumption
	starting in state w, using speed s at the first time step
$\hat{q}(w,s)$	Estimated discounted energy consumption
	starting in state w, using speed s at the first time step
(PL)	Synchronous learning of the probability matrix algorithm
(SQL)	Synchronous Qlearning Algorithm
(AQL)	Asynchronous Qlearning Algorithm
h_n	The n-th harmonic number:
	$h_n = \sum_{i=1}^n 1/i = \log(n) + \gamma + o(1/n)$ with γ the euler constant

Table 2.2.: Notations used throughout the thesis in particular in Chapter 6 and 7 (part 2).

Offline Minimization 3

Before analyzing the online case in the next chapters, we focus in a first time on the offline case. In this situation, the processor knows at each instant all the features (*i.e.* arrival times, execution times and deadlines) for past, active and future jobs. The goal is to minimize off-line the total energy consumption required to execute a set of n real-time jobs on a single processor with varying speed. We propose in this chapter a solution based on dynamic programming, that improves the time complexity of this problem. Let's start first with Section 3.1, where we present algorithms that has been done in the literature to address this problem, and their respective complexity.

This chapter is based on the paper published in the conference [GGP19b] and also on [GGP20a].

3.1 State of the Art

Several algorithms have been proposed in the literature to adapt processor speed dynamically by using DVFS techniques when the number of available speeds is finite, the algorithm with the best arithmetic complexity has been designed by Yao et al. [LYY17] and is in $\mathcal{O}(n\log(n))$, where n is the number of real-time jobs to schedule[1]. Based on statistical data on the job characteristics (arrival time, WCET, and deadline), Gaujal et al. [GGP17] have proposed a new approach based on a Markov Decision Process, that computes the optimal *on-line* speed scaling policy that minimizes the energy consumption on a single processor. In this chapter, we show that their algorithm can be adapted to the *off-line* case where the characteristics of the jobs are given as inputs to the algorithm. The analysis of this off-line version shows that its complexity is in $\mathcal{O}(n)$.

The problem of computing off-line DVFS schedules to minimize the energy consumption has been well studied in the literature, starting from the seminal paper of Yao et al. in 1995 [YDS95]. All the previous algorithms proposed in the literature compute the *critical interval* of the set of jobs[2], using more and more refined techniques to do so. This started in 1995 with [YDS95] and [Sta+98] where it was independently shown that one can compute the optimal speed schedule with complexity $\mathcal{O}(n^3)$, where n is the number of real-time jobs to schedule[3]. Later, [GN07] showed in 2007 that the complexity can be reduced to $\mathcal{O}(n^2L)$, where L is the nesting level of the set of jobs. Finally the complexity has been reduced to $\mathcal{O}(n^2)$ in the most recent work in 2017 [LYY17].

[1]The arithmetic complexity of an algorithm is the number of elementary operations it requires, regardless of the size of their arguments.
[2]The critical interval is the time interval with the highest load per time unit.
[3]The arithmetic complexity of an algorithm is the number of elementary operations it requires, regardless of the size of their arguments.

When the number of available speeds is *finite*, equal to m, [MCZ07] gave in 2007 a $\mathcal{O}(n^2)$ algorithm, while [LY05] proposed in 2005 a $\mathcal{O}(mn \log n)$ algorithm. In their most recent work, the same authors showed in 2017 that the complexity can be further reduced to $\mathcal{O}(n \log(\max\{m, n\}))$ [LYY17].

In this chapter, we present a *dynamic programming* solution that sweeps the set of jobs and computes the best speed at each time step while checking feasibility. The complexity is *linear* in the number of jobs, equal to Kn, where the constant K depends on the maximal speed and on a bound on the maximal relative deadlines of the jobs.

We recall briefly in Section 3.2 the system model, presented in Chapter 2. Then we detail in Section 3.3 the state space. Our dynamic programming solution is detailed in Section 3.4. We then study two extensions of our algorithm. First we show in Section 3.5.1 that the power of dynamic programming allows us to generalize this approach to the case where switching from one speed to another is not free, but instead takes some time ρ and may also have an energy cost h_e, which is a more realistic model. Second we show in Section 3.5.2 that our model allows for arbitrary power functions, not necessarily convex in the speed. Finally we provide concluding remarks in Section 3.6.

3.2 System Model

The model we consider is based on this one presented in Chapter 2. The particularity of this chapter is that as the job sequence is finite and all job features are known: arrival time, execution time and relative deadline for each jobs are known. As a consequence, the *time horizon* of the system is known and finite. As in Chapter 2, this *time horizon* is denoted by T and is the last deadline among all jobs of the system. It is defined as:

$$T = \max_{i=1}^{n}\{D_i\}. \tag{3.1}$$

The single core processor is equipped with m processing speeds also assumed to be in $\mathbb{N}$, and $s_{\max}$ denotes the maximal speed. The set of available speeds is denoted $\mathcal{S}$. The speeds are not necessarily consecutive integers. In the first part of the chapter, we assume the cost of speed switching to be null. This will be generalized in Section 3.5 for a non-null speed switching.

As explained in Chapter 2, we use the *Earliest Deadline First* (EDF) preemptive scheduling policy.

We recall that the power dissipated at any time t by the processor running at speed $s(t)$ is denoted $P_{ower}(s(t))$. For the time being, we assume that the P_{ower} function is *convex* (this assumption will be relaxed in Section 3.5.2). According to the notations of Chapter 2, the total energy consumption E is:

$$E = \int_1^T P_{ower}(s(t))\mathrm{d}t. \tag{3.2}$$

To sum up, given a set of n jobs $\{J_i\}_{i=1..n}$, the goal is to find an *optimal speed schedule* $\{s(t), t \in [1, T]\}$ that will allow the processor to execute all the jobs before their deadlines while minimizing the total energy consumption E.

3.3 State Space

3.3.1 State Description

The central idea of this chapter is to define the *state* of the system at time t. We denote $\mathcal{W}$ the set of all states of the system.

A natural state of the system at time t is the set of all jobs present at time t, i.e., $\{J_i = (r_i, c_i, d_i) | r_i \leq t \leq r_i + d_i\}$. Yet, in order to compute the speed of the processor, one does not need to know the set of actual jobs but only the *cumulative remaining work* present at time t, corresponding to these jobs. Therefore, a more compact state will be the *remaining work function* $w_t(.)$ at time t: for any $u \in \mathbb{R}^+$, $w_t(u)$ is the amount of work that must be executed before time $u + t$, taking into account all the jobs J_i present at time t (*i.e.* with a release time $r_i \leq t$ and deadline $r_i + d_i > t$). By definition, the remaining work $w_t(.)$ is a *staircase function*.

To derive a formula for $w_t(.)$, let us introduce the work quantity that arrives at any time t: to achieve this, we define in Def. 3.1 a function $a_t(.)$. For any $u \in \mathbb{R}^+$, the quantity $a_t(u)$ is the amount of work that arrives at time t and must be executed before time $t + u$.

Definition 3.1. *The amount of work that arrived at time t and must be executed before time* $t + u$ *is*

$$a_t(u) = \sum_{i \,|\, r_i = t} c_i H_{r_i + d_i}(t + u), \tag{3.3}$$

where $H_{d_i}(.)$ *is the discontinuous step function defined* $\forall x \in \mathbb{R}$ *as follows:*

$$H_{r_i+d_i}(x) = \begin{cases} 0 & \text{if } x < r_i + d_i, \\ 1 & \text{if } x \geq r_i + d_i. \end{cases} \tag{3.4}$$

To illustrate the definition of $a_t(.)$, let us consider an example with 3 jobs J_1, J_2, J_3 with respective release times $r_1 = r_2 = r_3 = t$, sizes $c_1 = 1$, $c_2 = 2$, $c_3 = 1$ and relative deadlines $d_1 = 2$, $d_2 = 3$, $d_3 = 5$. In this case, the function $a_t(.)$ is displayed in the middle graph of Fig. 3.1.

Def. (3.1) allows us to describe the state change formula when moving from time $t - 1$ to time t, using speed $s(u)$ in the while interval $[t - 1, t]$.

Lemma 3.1. *At time* $t \in \mathbb{N}$ *the remaining work function is given by:*

$$w_t(.) = \mathbb{T}\left[\left(w_{t-1}(.) - \int_{t-1}^{t} s(u)du\right)^+\right] + a_t(.), \tag{3.5}$$

with $\mathbb{T}f$ *the shift on the time axis of function* f*, defined as:* $\mathbb{T}f(t) = f(t+1)$ *for all* $t \in \mathbb{R}$*, and* $f^+ = \max(f, 0)$*, the positive part of a function* f*.*

Proof. Eq. (3.5) defines the evolution of the remaining work over time (see Fig. 3.1 for an illustration). The remaining work at time t is the remaining work at $t - 1$ minus the amount of work executed by the processor from $t - 1$ to t (which is exactly $\int_{t-1}^{t} s(u)du$) plus the work

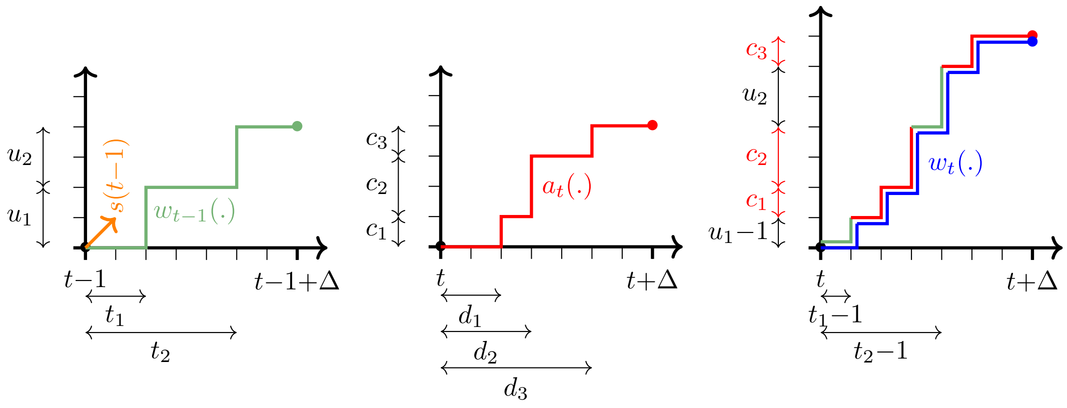


Figure 3.1.: **Left**: State of the system at $t-1$. The green line depicts the remaining work function $w_{t-1}(.)$. The constant speed chosen between times $t-1$ and t is $s(t-1)=1$; u_1 stands for $w_{t-1}(2)$ and u_2 stands for $w_{t-1}(5)-w_{t-1}(2)$. **Middle**: Arrival of three new jobs (r_i, c_i, d_i) at t: $J_1 = (t,1,2)$, $J_2 = (t,2,3)$, and $J_3 = (t,1,5)$. The red line depicts the corresponding arrival work function $a_t(.)$. **Right**: The blue line depicts the resulting state at t, $w_t(.)$, obtained by shifting the time from $t-1$ to t, by executing 1 unit of work (because $s(t-1)=1$), and by incorporating the jobs arrived at t. Above the blue line are shown in green the "parts" of $w_t(.)$ that come from $w_{t-1}(.)$ and in red those from $a_t(.)$.

arriving at t. The "max" with 0 makes sure that the remaining work is always positive and the $\mathbb{T}$ operation performs a shift of the reference time from $t-1$ to t. □

3.3.2 Size of the State Space

As said in Section 3.3.1, $\mathcal{W}$ is the set of all states of the system. $\mathcal{W}$ is therefore the set of all possible remaining work functions that can be reached by any *feasible* set of jobs, when the processor only changes its speed at integer times, and when no job has missed its deadline before time t. The size of the state space $\mathcal{W}$ is denoted by Q.

As explained in Appendix A.2, the size Q of the state space $\mathcal{W}$ can be computed using a generalization of the Catalan numbers. The number of Catalan paths from $(0,0)$ to $(\Delta+1), s_{\max}(\Delta+1))$, hence the number of all possible remaining work functions for any set of schedulable jobs, is:

$$Q = \frac{1}{1+s_{\max}(\Delta+1)}\binom{(s_{\max}+1)(\Delta+1)}{\Delta+1} \approx \frac{e}{\sqrt{2\pi}}\frac{1}{(\Delta+1)^{3/2}}(e\,s_{\max})^{\Delta}. \tag{3.6}$$

3.4 Dynamic Programming Solution

The goal of this section is to describe a dynamic program that computes an optimal speed schedule $s^*(t), t \in [1,T]$, such that $s^*(t)$ minimizes the energy consumption among all schedules where the speed may only change at integer times (the speed is therefore a piece-wise constant function). We distinguish the case where the speeds form a consecutive set (that is, $\mathcal{S} = \{0,1,\ldots,m-1\}$) and the case where they do not.

3.4.1 Consecutive Speeds

Algorithm 1 computes the optimal speed schedule. Before presenting its pseudo-code, let us provide an informal description of the behavior of the system. Under a given piece-wise constant speed schedule $s(1), s(2), \ldots, s(T-1)$, the state of the system evolves as follows:

- At time 0, no jobs are present in the system so the initial state function w_0 is the null function, which we represent by the null vector of size Δ: $w_0 = (0, \ldots, 0)$ (see Line 4).

- The first job J_1 is released at time 1, maybe simultaneously with other jobs, so the new state function becomes $w_1 = w_0 + a_1$ according to Eq. (3.5). The case where several jobs are released at time 1 is taken care by the sum operator in Eq. (3.3) used to compute a_1.

- At time 1, the speed of the processor is set to $s(1)$. The processor uses this speed up to time 2, incurring an energy consumption equal to $P_{ower}(s(1))$.

- At time 2, the state function becomes $w_2 = \mathbb{T}(w_1 - s(1))^+ + a_2$ according to Eqs. (3.5) and (3.3), and so on and so forth up to time $T-1$, resulting in the sequence of state functions $w_1, w_2, \ldots, w_{T-1}$.

Now, let us denote by $E_t^*(w)$ the minimal energy consumption from time t to time T, if the state at time t is w, and if the optimal speed schedule is $s^*(t), s^*(t+1), \ldots, s^*(T-1)$. Of course, this partial optimal schedule is not known. But let us assume (using a backward induction) that the optimal speed schedule is actually known *for all possible states* $w \in \mathcal{W}$ at time t. It then becomes possible to compute the optimal speed schedule for all possible states between time $t-1$ and T using the maximum principle:

$$
\begin{aligned}
E_{t-1}^*(w) &= \min_{s \in \mathcal{S}} \left(P_{ower}(s) + E_t^*(\mathbb{T}(w-s)^+ + a_t) \right) && (3.7)\\
s^*(t-1)(w) &= \operatorname*{argmin}_{s \in \mathcal{S}} \left(P_{ower}(s) + E_t^*(\mathbb{T}(w-s)^+ + a_t) \right), && (3.8)
\end{aligned}
$$

where $s^*(t)(w)$ denotes the optimal speed at time t if the current state is w.

When time 0 is reached, the optimal speed schedule has been computed between 0 and T *for all possible initial states*. To obtain an optimal speed schedule for the sequence of states $w_1, \ldots, w_{T-1}$, we just have to return the speeds $s^*(1)(w_1), \ldots, s^*(T-1)(w_{T-1})$ (see Line 26). Note that, because of the "argmin" operator in Eq. (3.8), the optimal speed schedule is not necessarily unique.

This is what Algorithm 1 below does. E^* is computed using the backward induction described previously, which is a special case of the finite horizon policy evaluation algorithm provided in [Put05] (p. 80).

The cases where the set of jobs is not schedulable are taken into account by setting the energy function $E_t^*(w')$ to infinity if the state w' is not schedulable, that is, if $w' \notin \mathcal{W}$ (see lines 12 and 13) since $\mathcal{W}$ is the set of feasible states by definition.

Algorithm 1: Dynamic programming algorithm computing the optimal speed schedule.

```
1:  input: $\{J_i = (\tau_i, c_i, d_i), i = 1..n\}$                % Set of jobs to schedule
                                                                   % Initializations
2:  for all $i = 1$ to $n$ do $r_i \leftarrow \sum_{k=0}^{i} \tau_k$ end for      % Release times
3:  $T \leftarrow \max_i(r_i + d_i)$                                % Time horizon
4:  $w_0 \leftarrow (0, \dots, 0)$
5:  for all $\boldsymbol{x} \in \mathcal{W}$ do $E_T^*(\boldsymbol{x}) \leftarrow 0$ end for     % Energy at the horizon
                                                                   % Main loop
6:  $t \leftarrow T$                                                % Start at the horizon
7:  while $t \geq 1$ do
8:      for all $\boldsymbol{x} \in \mathcal{W}$ do
9:          $E_{t-1}^*(\boldsymbol{x}) \leftarrow +\infty$
10:         for all $s \in (\mathrm{MP})(\boldsymbol{x})$ do
11:             $\boldsymbol{y} \leftarrow \mathbb{T}\,[(\boldsymbol{x} - s)^+] + a_t$      % Computation of the next state
12:             if $\boldsymbol{y} \notin \mathcal{W}$ then
13:                 $E_t^*(\boldsymbol{y}) \leftarrow +\infty$              % The next state is unfeasible
14:             end if
15:             if $E_{t-1}^*(\boldsymbol{x}) > P_{ower}(s) + E_t^*(\boldsymbol{y})$ then
16:                 $E_{t-1}^*(\boldsymbol{x}) \leftarrow P_{ower}(s) + E_t^*(\boldsymbol{y})$     % Update the energy in state $\boldsymbol{x}$ at $t-1$
17:                 $s^*(t-1)(\boldsymbol{x}) \leftarrow s$     % Update the optimal speed in state $\boldsymbol{x}$ at $t-1$
18:             end if
19:         end for
20:     end for
21:     $t \leftarrow t - 1$                                        % Backward computation
22: end while
23: if $E_1^*(\boldsymbol{x}_1) = +\infty$ then                      % Return the result
24:     return "not feasible"
25: else
26:     return $\{s^*(t)(\boldsymbol{x}_t)\}_{t=1\dots T}$
27: end if
```

If $s(t)(\boldsymbol{x})$ is the speed that the processor has to use at time t in state $\boldsymbol{x}$, then the deadline constraint on the jobs imposes that $s(t)(\boldsymbol{x})$ must be large enough to execute the remaining work at the next time step, and cannot exceed the total work present at time t. This means:

$$\forall t, \forall \boldsymbol{x}, \quad w(\Delta) \geq s(t)(\boldsymbol{x}) \geq \boldsymbol{x}(1). \tag{3.9}$$

This set of *admissible* speeds in state $\boldsymbol{x}$ will be denoted by $(\mathrm{MP})(\boldsymbol{x})$ and formally defined as:

$$(\mathrm{MP})(\boldsymbol{x}) = \big\{ s \in \mathcal{S} \text{ s.t. } w(\Delta) \geq s \geq w(1) \big\}. \tag{3.10}$$

Our first result is Theorem 3.1, which states that Algorithm 1 computes the optimal speed schedule.

Theorem 3.1. *Assume that the speeds form a consecutive set,* i.e. $\mathcal{S} = \{0, 1, \dots, m-1\}$. *If the set of jobs is not schedulable, then Algorithm 1 outputs "not schedulable". Otherwise it outputs an optimal speed schedule that minimizes the total energy consumption.*

Proof. **Case A: The set of jobs is not schedulable.** Then, at some time t, the state $\boldsymbol{x}_t$ will get out of the set of schedulable states, for all possible choices of speeds. Hence its value $E_t^*(\boldsymbol{x}_t)$ will be set to infinity (see Line 13) and this will propagate back to time 1. In conclusion, $E_1^*(\boldsymbol{x}_1)$ will be infinite and Algorithm 1 will return "not schedulable" (see Line 24).

Case B: The set of jobs is schedulable. The proof proceeds in two stages. In the first stage we show that there exists an optimal solution where speed changes only occur at integer times. In the second stage, we show that Algorithm 1 finds an optimal speed schedule among all solutions that only allow speed changes at integer times.

Case B – first stage. To prove that there exists an optimal solution where speed changes only occur at integer times, let us first present the algorithm that computes the optimal speed schedule described in [YDS95]. The core principle of this algorithm is to compute the *critical interval*, denoted I^c and defined as the time interval with the highest average amount of work. In general, there can be several such intervals, in which case we pick anyone.

To formally define of the critical interval, we rely on the release time r_i of job J_i (defined in Eq. (2.1)) and on its absolute deadline D_i (defined in Section 2.1). We say that a job J_i *belongs* to an interval $I = [u, v]$, denoted $J_i \in [u, v]$, if and only if $r_i \geq u$ and $D_i \leq v$. Using this notation, the critical interval I^c is:

$$I^c = [u^c, v^c] = \underset{I=[u,v]}{\operatorname{argmax}} \frac{\sum_{J_i \in I} c_i}{v - u}. \tag{3.11}$$

Let ℓ be the length of I^c: $\ell = v^c - u^c$. Furthermore, let ω be the total amount of work in I^c: $\omega = \sum_{J_i \in I^c} c_i$. Since the power is a convex function of the speed, the optimal speed s^c over I^c is *constant* and equal to $s^c = \frac{\omega}{\ell}$.

When the set $\mathcal{S}$ of available speeds is finite, the optimal solution is inferred from the unconstrained case as follows: Pick the two *neighboring* available speeds s_1 and s_2 in $\mathcal{S}$ such that s^c belongs to $[s_1, s_2)$. As a consequence, s^c is equal to a linear combination of s_1 and s_2:

$$s^c = \alpha s_1 + (1-\alpha)s_2, \text{ with } \alpha = \frac{s_2 - s^c}{s_2 - s_1}. \tag{3.12}$$

An optimal speed schedule over the critical interval I^c is therefore obtained by selecting speed s_1, possibly over several sub-intervals, for a cumulative time equal to $\alpha\ell$, and speed s_2 for a total time equal to $(1-\alpha)\ell$ (the rest of the interval).

The first critical interval I^c is computed over the working interval $[1, T]$ and the initial set of jobs $\{J_i\}_{i=1..n}$. Then, I^c is *collapsed*, meaning that:

- the new working interval $[1, T]$ becomes the union $[1, u^c] \cup [v^c, T]$, called the contracted interval;
- and all the jobs included in I^c are removed from the set of jobs to be scheduled.

The new critical interval is then constructed over the contracted interval and over the remaining jobs, and so on and so forth.

This ends the description of the optimal solution and we are now ready for the proof that there exists an optimal solution whose speed changes occur at integer times.

Since the release times and deadlines are integers, the critical interval I^c has integer bounds: $I^c = [u^c, u^c + \ell]$ with $u^c, \ell \in \mathbb{N}$. Since the sizes of the jobs are also integer, the total amount of work over I^c is also integer: $\omega \in \mathbb{N}$.

When the set of available speeds is consecutive, $\mathcal{S} = \{0, 1, \dots, m-1\}$, the two neighboring available speeds of $s^c = \frac{\omega}{\ell}$ satisfy $s_1 \le s^c < s_2 = s_1 + 1$. In this case,

$$s^c = \frac{\omega}{\ell} = \alpha s_1 + (1-\alpha)(s_1+1) = s_1 + 1 - \alpha. \tag{3.13}$$

This implies that $\alpha\ell = \ell(s_1+1) - \omega$. Since $\ell, \omega, s_1 \in \mathbb{N}$, then $\alpha\ell \in \mathbb{N}$. This means that the speeds s_1 and s_2 will both be used during an integer amount of time. One optimal speed schedule can be constructed by using speed s_1 over $\alpha\ell$ intervals of length one, and speed $s_1 + 1$ over $(1-\alpha)\ell$ intervals of size one, constructed in the following manner: The speed $s^*(k) \in \{s_1, s_1+1\}$ used in interval $[u^c + k - 1, u^c + k]$, for $k = 1..\ell$, is:

$$s^*(k) = \lfloor ks^c \rfloor - \lfloor (k-1)s^c \rfloor. \tag{3.14}$$

This choice of speeds makes sure that speed s_1 is used during $\alpha\ell$ unit intervals and speed s_2 during $(1-\alpha)\ell$ unit intervals over the critical interval. In addition, under speeds $s^*(k)$, the jobs in the critical interval are all executed within their deadlines because of the following two reasons:

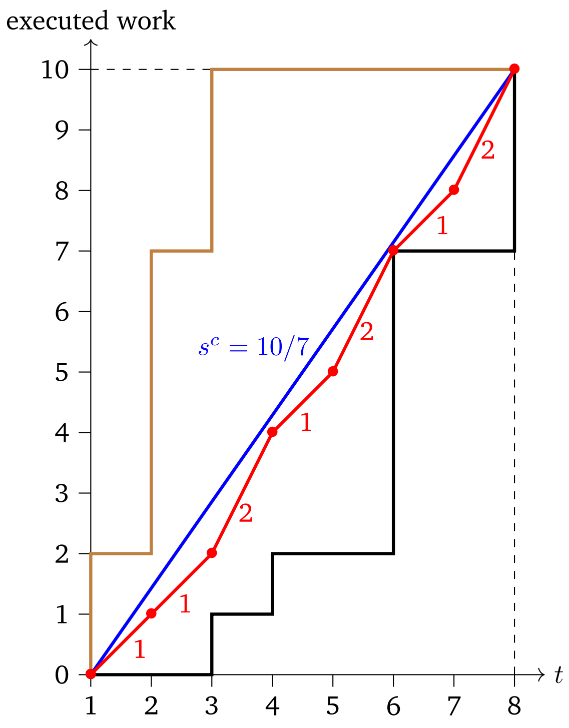


Figure 3.2.: Construction of the optimal solution over a critical interval made of 4 jobs $\{J_i\}_{i=1..4}$ whose features $(r_i, c_i, D_i)_{i=1..4}$ are $(1,1,3)$, $(1,1,4)$, $(2,5,6)$, and $(3,3,8)$. The corresponding cumulative deadlines form the black staircase while the cumulative arrivals form the brown staircase. The critical interval is $I^c = [1,8]$, the total amount of work over I^c is $\omega = 10$, the optimal speed is $s^c = 10/7$, and its neighboring speeds are $s_1 = 1$ and $s_2 = s_1 + 1 = 2$. The optimal speed schedule only uses speeds 1 and 2 and only changes speeds at integer times. The sequence of optimal speeds given by Eq. (3.14) is $(1,1,2,1,2,1,2)$.

1. On the one hand, for any k strictly less than v^c, the sizes of the jobs in the critical interval with deadlines smaller than $k + u^c$ must sum up to a value V_k not larger than ks^c by non-criticality of the sub-interval $[u^c, u^c + k]$. Since the sizes of the jobs are integers, one further gets $V_k \leq \lfloor ks^c \rfloor$.

2. On the other hand, Eq. (3.14) implies that $s^*(1) + s^*(2) + ... + s^*(k) = \lfloor ks^c \rfloor$.

As a consequence, $V_k \leq s^*(1) + s^*(2) + ... + s^*(k)$, meaning that the sum of the sizes of the jobs belonging to the interval $[u^c, u^c + k]$ is less than the total amount of the work executed by the processor during the interval $[u^c, u^c + k]$.

Under this optimal solution, all the speed changes occur at integer points. The construction of this optimal solution is illustrated in Fig. 3.2: The integer cumulative deadlines (black staircase) are below the straight line whose slope is s^c (blue line) if and only they are also below the broken line with slopes $s^*(k)$ (red broken line).

Case B – second stage. In the second stage of the proof, we show that Algorithm 1 finds an optimal speed selection among all solutions that only allow speed changes at integer times. Together with the first stage, this will end the proof. Proving the optimality of Algorithm 1 is classical in dynamic programming. This is done by a backward induction on the time t. Let us show that $E_t^*(\boldsymbol{x})$, as computed by the algorithm, is the optimal energy consumption from time t to time T under any possible state $\boldsymbol{x}$ at time t.

Initial step: $t = T$. We set $E_T^*(\boldsymbol{x}) = 0$ for all w. Indeed no jobs are present after time T, so that the state reached at time t must be $w_T = (0, 0, \dots, 0)$ because no work is left at time T and the value $E_T^*(\boldsymbol{x}_T) = 0$ is therefore correct. No speed has to be chosen at time T.

Induction: Assume that the property is true at time $t+1$. At time t, Algorithm 1 computes $\forall \boldsymbol{x} \in \mathcal{W}$, $E_t^*(\boldsymbol{x})$. In particular, if the set of jobs is schedulable, then the actual state at time t, w_t, must be in $\mathcal{W}$. Therefore, according to Lines 15 and 16, we have:

$$E_t^*(\boldsymbol{x}_t) = \min_{s \in (\mathrm{MP})(\boldsymbol{x})} \left(P_{ower}(s) + E_{t+1}^*(\mathbb{T}(\boldsymbol{x}_t - s)^+ + a_t) \right).$$

All possible speeds at time t are tested with their optimal continuation (by induction hypothesis). Therefore, the best choice of speed at t, which minimizes the total energy from t to T, is selected by Algorithm 1.

Finally, when all the speed changes occur at integer times, the total energy consumption computed by Eq. (3.2) is equal to the value $E_1^*(\boldsymbol{x}_1)$ computed by Algorithm 1. □

Our second result is Theorem 3.2, which states that the time complexity of Algorithm 1 is linear in the number of jobs n.

Theorem 3.2. *The time complexity of Algorithm 1 is Kn, where n is the number of jobs and the constant K depends on the maximal speed $s_{\max}$ and the maximal relative deadline Δ.*

Proof. The proof proceeds by inspecting Algorithm 1 line by line. The number of operations in Line 11 is equal to the number of jobs whose release time is at time t, denoted n_t:

$$n_t = |\{J_i = (r_i, c_i, d_i) \text{ s.t. } r_i = t\}|, \tag{3.15}$$

and the sum of all n_t is equal to the total number of jobs, n:

$$\sum_{t=1}^{T} n_t = n. \tag{3.16}$$

Furthermore, the number of operations in Line 12 is Δ (to check if $w'(i) \le i\, s_{\max}$ for $i = 1..\Delta$). Therefore the total number O of arithmetic operations (copies, comparisons and additions of integers) is:

$$O = \sum_{t=1}^{T} \sum_{\boldsymbol{x} \in \mathcal{W}} \sum_{s \in (\text{MP})(\boldsymbol{x})} (n_t + \Delta + K'), \tag{3.17}$$

where K' is a constant.

The size of $(\text{MP})(\boldsymbol{x})$ is bounded by $s_{\max}$[4]. Hence O is bounded by a linear function of n and T:

$$O \le nQs_{\max} + TQs_{\max}(\Delta + K'). \tag{3.18}$$

We have seen previously that Q is bounded by a function of $s_{\max}$ and Δ (see Appendix A.2). Now, $T = \max_{i=1}^{n}(r_i + d_i) = \max_{i=1}^{n}(d_i + \sum_{j=1}^{i} \tau_j)$. If there exists j such that $\tau_j > \Delta$, then there exists an interval of time when the processor must be idle, between the end of the execution of the first $j-1$ jobs and the release time of the j^{th} job. In this case the problem can be split into two: all jobs from 1 to $j-1$ and all jobs from j to n.

This means that one can assume with no loss of generality that all inter-arrival times are smaller than Δ, hence $T \le n\Delta$.

It follows, the total number of arithmetic operations O is bounded:

$$O \le nK \quad \text{with} \quad K = Qs_{\max}(\Delta^2 + \Delta K' + 1). \tag{3.19}$$

Finally, by replacing in Eq. (3.19) Q by its value from Eq. (3.6), we conclude that exists a constant K_0 such that

$$O \le n \times K_0 \sqrt{\Delta} (e\, s_{\max})^{\Delta+1}. \tag{3.20}$$

□

[4] To be more precise, $|(\text{MP})(\boldsymbol{x})|$ is bounded by $|\mathcal{S}|$, and since $\mathcal{S} = \{0, 1, \dots m-1\}$, we have $|\mathcal{S}| = s_{\max} + 1$.

3.4.2 Reducing the Complexity

The main term in Eq. (3.19) is Q, the size of the state space $\mathcal{W}$. The dynamic programming algorithm 1 computes the optimal energy for all states in Q at each time t, regardless of the fact that these states are reachable at time t.

We present in this section an improved algorithm that constructs the set of reachable states *on the fly* at each time step t, resulting in a dramatic reduction of the complexity, from $\mathcal{O}(n \times s_{\max}\sqrt{\Delta}(e\, s_{\max})^{\Delta+1})$ to $\mathcal{O}(n \times (s_{\max}C\Delta^2))$.

First, let us consider the cumulative evolution up to time t. Let $e(t)$ be the work executed up to time t:

$$e(t) = \sum_{i=0}^{t} s(i), \tag{3.21}$$

where $s(i)$ denotes the speed used at time i. The executed work $e(t)$ must be smaller than the cumulative work $A(t)$ arrived before time t, and larger than the cumulative deadlines $D(t)$ at t:

$$D(t) \le e(t) \le A(t),$$

with

$$A(t) = \sum_{i:r_i \le t} c_i \qquad \text{and} \qquad D(t) = \sum_{i:D_i \le t} c_i. \tag{3.22}$$

At time 0, $A(0) = e(0) = D(0) = 0$ and at time T, $A(T) = e(T) = D(T) = \sum_{i=1}^{n} c_i$.

As discussed earlier, feasibility implies that at the backlog cannot become greater than $s_{\max}\Delta$. Therefore, under a schedulable set of jobs, we have $A(t) - D(t) = \sum_{i:r_t \le t < D_i} c_i \le s_{\max}\Delta$, hence for any t the number of different values for $e(t)$ is smaller than $s_{\max}\Delta$.

To refine the bounds on $e(t)$, we define $M(t)$ as the maximal amount of executed work:

$$M(t) = \min\big(A(t), M(t-1) + s_{\max}\big) \quad \text{with} \quad M(0) = 0. \tag{3.23}$$

At time t, the maximal amount of executed work $M(t)$ can be bounded by $A(t)$ as discussed above, but also by $M(t-1) + s_{\max}$. This means that at any time t we have

$$D(t) \le e(t) \le M(t).$$

Second, the state at time t is a function of $e(t)$. If we denote by $w_t^{e(t)}(.)$ the work remaining function at time t when a quantity $e(t)$ of work has been executed up to time t, then, for all $u \ge 0$, we have:

$$w_t^{e(t)}(u) = \left(\sum_{i:r_i<t} c_i H_{r_i+d_i}(t+u) - e(t)\right)^+ + \sum_{i:r_i=t} c_i H_{r_i+d_i}(t+u). \tag{3.24}$$

In other words, $w_t^{e(t)}(.)$ is a function of $e(t)$. Since there are $s_{\max}\Delta$ different values of $e(t)$, the same holds for $w_t(.)$. As a result, the number of reachable states at time t is smaller than $s_{\max}\Delta$.

Finally, to make the construction of all reachable states more efficient, the dynamic programming should be done in a *forward* mode, instead of backward as it is done in Algorithm 1, because this allows us to construct the state associated to $e(t)$ incrementally and iteratively, by using the states at time $t-1$. The resulting forward algorithm is shown in Algorithm 2.

Theorem 3.3. *Algorithm 2 computes the optimal speed schedule using less than $n \times K s_{\max}^2 \Delta^3$ operations, where K is a constant.*

Proof. Let us decompose the analysis of the complexity step by step:

- To compute the cumulative functions $A(t)$ and $D(t)$ in Lines 8 and 9, the complexity is $\sum_{t=1}^{T}(n_t + n_t + K_1) \leq 2n + K_1 T$, where n_t is the number of tasks released at time t and K_1 is a constant.
- In the main temporal loop, from line (20) to (33), there are three parts:
  1. At Line 21, the complexity of the energy initialization is bounded by $K_2 s_{\max}\Delta$, where K_2 is a constant.
  2. From line 22 to 33, there are 2 nested loops, one on e', bounded by $s_{\max}\Delta$ and one another on speed s, bounded by $s_{\max}$. In this part, we use the minimum principle to determine the minimal energy. All these computations are done in constant time except for line 30. Therefore, the time complexity of the rest is bounded by $K_3 s_{\max}^2 \Delta$, where K_3 is a constant.
  3. In line 30, the state associated to e at time t is constructed. It takes at most Δ operations to subtract s and take the positive part. Moreover n_t additions are needed to add the sizes of the new jobs arrived at t. As a result, the time complexity here is bounded by $K_4 s_{\max}^2 \Delta(\Delta + n_t)$, where K_4 is a constant.

The whole loop has a complexity $K_5(s_{\max}^2 \Delta T + s_{\max}^2 \Delta^2 T + s_{\max}^2 \Delta n_t)$, where K_5 is a constant.

Moreover, the result output (line 38) uses $K_5 T$ operations.

Finally, $T \leq n\Delta$ (see the proof of Theorem 3.2).

Putting everything together yields a number of elementary operations (copies of an integer, comparisons, additions) bounded by $n \times K s_{\max}^2 \Delta^3$, where K is a constant that does not depend on the problem instance.. □

Figure 3.3 displays all states visited by Algorithm 2 with the set of jobs given at the left of the figure and with the set of available speeds $\{0, 1, 2\}$. The speeds considered in each state for optimizing the energy are shown as black arrows. Note that speed 0 is not considered between times 5 and 6. This is because for any point e at time 5, we have $w_5^e(1) = 1$. This value comes from Job J_3 that arrives at time 5 with a relative deadline of 1. Also note that the point $e = 5$ at time 5 (the blue cross in Fig. 3.3) is not visited because it is below $M(5) = 4$.

The two following corollaries are the main result of this chapter.

Corollary 1. *Algorithm 2 can be improved in order to compute the optimal speed schedule and use less than $n \times K s_{\max}^2 \Delta^2$ operations, where K is a constant.*

Algorithm 2: Optimized dynamic programming algorithm computing the optimal speeds.

```
1:  input: $\{J_i = (\tau_i, c_i, d_i), i = 1..n\}$                         % Set of jobs to schedule
                                                                             % Initializations
2:  for all $i = 1$ to $n$ do $r_i \leftarrow \sum_{k=0}^{i} \tau_k$ end for          % Release times
3:  $T \leftarrow \max_i(r_i + d_i)$                                          % Time horizon
4:  $w_0^0 \leftarrow (0, \dots, 0)$
5:  $A(0) \leftarrow 0$; $D(0) \leftarrow 0$; $M(0) \leftarrow 0$
6:  for all $t = 1$ to $T$ do
7:      for all $i$ s.t. $r_i = t$ do
8:          $A(t) \leftarrow A(t) + c_i$                                       % arrivals at $t$
9:          $D(t + d_i) \leftarrow D(t + d_i) + c_i$                           % Deadlines
10:     end for
11: end for
12: for all $t = 1$ to $T$ do
13:     $A(t) \leftarrow A(t-1) + A(t)$                                        % Cumulative arrival staircase
14:     $D(t) \leftarrow D(t-1) + D(t)$                                        % Cumulative deadline staircase
15:     $M(t) \leftarrow \min(A(t), M(t-1) + s_{\max})$                        % Maximal executed work
16:     if $A(t) - D(t) > s_{\max}\Delta$ then
17:         return "not feasible"
18:     end if
19: end for
                                                                             % Main loop
20: for all $t = 1$ to $T$ do                                                % Forward computation
21:     for all $e \in [D(t), M(t)]$ do $E_t^*(e) \leftarrow +\infty$ end for     % Energy at each reachable state
22:     for all $e' \in [D(t-1), M(t-1)]$ do
23:         for all $s \in [w_{t-1}^{e'}(1), \min(s_{\max}, M(t) - e')]$ do % Sweep admissible speeds from $t{-}1$ to $t$
24:             $e \leftarrow e' + s$                                          % Amount of executed work at time $t$
25:             if $E_t^*(e) > P_{ower}(s) + E_{t-1}^*(e')$ then
26:                 $E_t^*(e) \leftarrow P_{ower}(s) + E_{t-1}^*(e')$          % Forward optimality equation
27:                 $s_t^*(e) \leftarrow s$
28:                 $prev_t^*(e) \leftarrow e'$                                % Store the optimal solution backwards
29:             end if
30:             $w_t^e \leftarrow \mathbb{T}(w_{t-1}^{e'} - s)^+ + a_t$          % Build the state associated to $e$ at $t$
31:         end for
32:     end for
33: end for
                                                                             % Return the result
34: if $E_T^*(e_T) = +\infty$ then
35:     return "not feasible"
36: else
37:     for all $t$ from $T$ to $1$ do                                       % Output the optimal solution backward
38:         return $s_t^*(e_t^*)$
39:         $e_{t-1}^* \leftarrow prev_t^*(e_t^*)$
40:     end for
41: end if
```

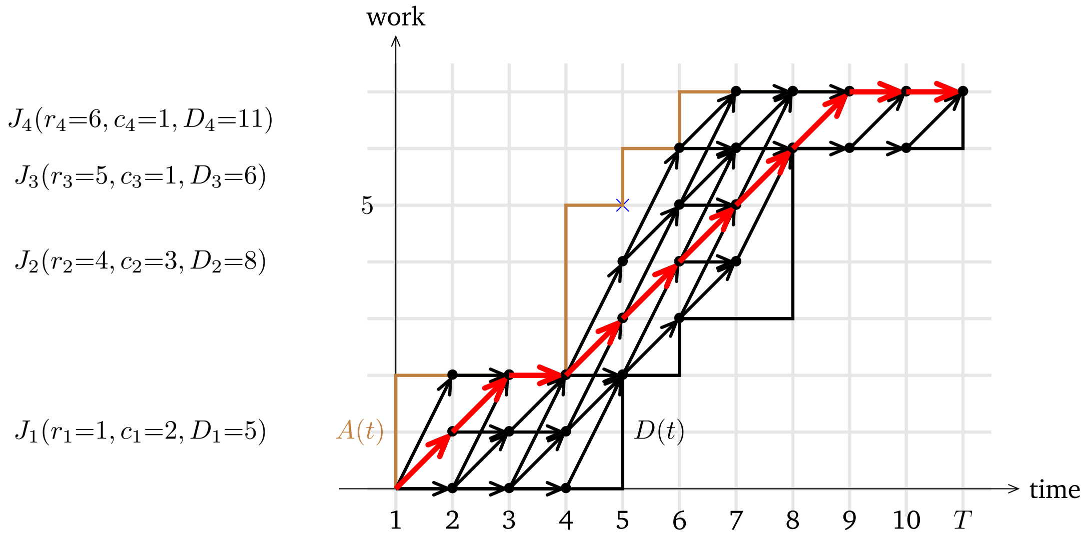

work
$J_4(r_4{=}6, c_4{=}1, D_4{=}11)$
$J_3(r_3{=}5, c_3{=}1, D_3{=}6)$
$J_2(r_2{=}4, c_2{=}3, D_2{=}8)$
$J_1(r_1{=}1, c_1{=}2, D_1{=}5)$
5
$A(t)$
$D(t)$
time
1 2 3 4 5 6 7 8 9 10 T

Figure 3.3.: Execution of Algorithm 2 with 4 jobs $\{J_i\}_{1=1..4}$. The cumulative deadlines form the black staircase $D(t)$, while the cumulative arrivals form the brown staircase $A(t)$. All the states visited by the algorithm are depicted as dots, and all the speeds evaluated in these states are depicted as arrows. The optimal speed schedule computed by Algorithm 2 is shown as the bold red arrows: $(1, 1, 0, 1, 1, 1, 1, 1, 0, 0)$.

Proof. The reduction from Δ^3 to Δ^2 can be achieved by replacing the state construction in line 30 in Algorithm 2 by the following code:

```
1: if $e \le M(t-1)$ and $s = 0$ then
2:     $w_t^e \underset{in-place}{\longleftarrow} \mathbb{T}(w_{t-1}^{e'}) + a_t$
3: else if $M(t-1) < e \le M(t)$ and $s = s_{\max}$ then
4:     $w_t^e \leftarrow \mathbb{T}(w_{t-1}^{e'} - s)^+ + a_t$
5: end if
```

Indeed, line 2 in the above code changes the vector $w_{t-1}^{e'}$ in place (symbol "$\underset{in-place}{\longleftarrow}$"), *i.e.* we only move a pointer position for the time shift operation $\mathbb{T}$(this can be done in constant time) and add a_t with cost $K_1 n_t$. Therefore, this line costs $K_6 n_t$ and will be visited $s_{\max}\Delta$ times at most.

As for line 4 in the above code, the copy of Δ values and the computation of the max cost $K_7\Delta$. However, this line will be visited only $s_{\max}$ times and not for all states. So the complexity of the state construction is reduced to $K_8 s_{\max}(\Delta + n_t)$.

Therefore the complexity of this replacement of line 30 in Algorithm 2 becomes: $K_9(s_{\max}\Delta n_t + s_{\max}(\Delta + n_t))$.

By adding the other terms computed in the proof of Theorem 3.3 and the temporal loop, we obtain a complexity of $K_{10}(2n + T + s_{\max}^2\Delta T + s_{\max}\Delta n + s_{\max}\Delta T + s_{\max}\Delta n)$.

□

Corollary 2. *If the work arriving at any instant t is bounded by C (*i.e.* $\forall t, \sum_{i:r_i=t} c_i \le C$), then Algorithm 2 computes the optimal speed schedule using less than $n \times K s_{\max} C\Delta^2$ operations.*

Proof. The complexity change from $s_{\max}^2$ to $s_{\max}C$ comes from the fact that $A(t) - D(t) \leq C\Delta$ if the work arriving at t is bounded by C. □

3.4.3 Non Consecutive Speeds

When the available speeds do not form a consecutive set, Algorithm 2 may not find an optimal schedule because it is possible that all the optimal speed schedules change speed at non integer times. Such solutions will not be found by Algorithm 2. In this section we show that it is possible to go back to the consecutive case by interpolating the power function.

Let $\mathcal{S}$ be the set of available speeds. First, we assign to each non available speed a power consumption by using a linear interpolation. Let $s \in \mathbb{N}$ such that $s < s_{\max}$ and $s \notin \mathcal{S}$. Let $s_1, s_2 \in \mathcal{S}$ be the two neighboring available speeds such that $s_1 < s < s_2$. It follows that s is a linear combination of s_1 and s_2:

$$s = \beta s_1 + (1 - \beta)s_2, \text{ with } \beta = \frac{s_2 - s}{s_2 - s_1}. \tag{3.25}$$

We therefore set:

$$P_{ower}(s) = \beta P_{ower}(s_1) + (1 - \beta)P_{ower}(s_2). \tag{3.26}$$

Once this is done for each non available speed, we use Algorithm 2 to solve the problem with all consecutive speeds between 0 and $s_{\max}$, the unavailable speeds being seen as available with the power cost defined in Eq. (3.26).

Once the optimal speeds have been computed, the following transformation is done at each time step. In the time interval $[t, t+1)$, if the optimal speed $s^*(t)$ was not originally available, then it is replaced by its two neighboring available speeds s_1 and s_2 over time intervals $[t, t+\beta)$ and $[t+\beta, t+1)$ respectively. This transformation is illustrated in Fig. 3.4. Since the deadlines are integers, no job will miss its deadline during the interval $(t, t+1)$.

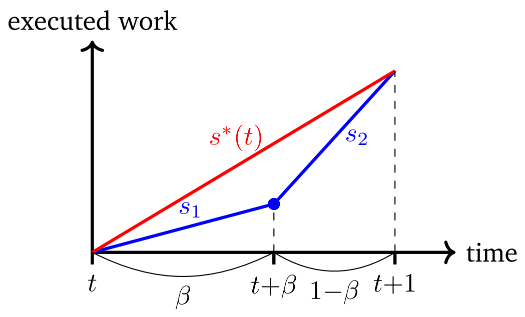


Figure 3.4.: Amount of work executed with speed $s^*(t)$ (in red), and amount of work executed by the two neighboring available speeds s_1 and s_2 (in blue).

Theorem 3.4 is the main result of this chapter.

Theorem 3.4. *When the set of available speeds is arbitrary, the improved Algorithm 2 together with the interpolation transformation displayed in Fig. 3.4 computes an optimal speed schedule to execute n jobs in time less than $n \times K s_{\max}^2 \Delta^2$ (or $n \times K s_{\max} C \Delta^2$ if the work arriving at any instant t is bounded by C), where K is a constant.*

Proof. Since the power cost $P_{ower}(s^*(t))$ is a linear interpolation of the power cost of the neighboring available speeds s_1 and s_2, the energy consumption over the interval $[t, t+1)$ is the same using speed s_t^* and using the neighboring speeds s_1 and s_2 over the two sub intervals $[t, t+\beta]$ and $[t+\beta, t+1]$. This means that the total energy consumption is the same before and after the transformation of Figure 3.4.

On the one hand, Theorem 3.1 states that, with consecutive speeds, the output of Algorithm 2 minimizes the total energy consumption.

We have just shown that the transformation of each speed into a convex combination between two neighboring speeds provides a solution that only uses the available speeds and has the same total energy consumption as with consecutive speeds.

On the other hand, the optimal solution only using the subset composed by the available speeds must use at least as much energy as when all the intermediate speeds are available. This implies that Algorithm 2, used with the interpolated power function where unavailable speeds are replaced by their neighboring available speeds, gives an optimal solution that minimizes the total energy consumption.

Finally, this transformation takes a constant amount of time for each time interval $[t, t+1)$, therefore, the complexity remains linear, with possibly a new constant. □

3.4.4 Comparison with Previous Work

If we want to compare our algorithm with the best algorithm presented in [LYY17] whose complexity is $K'' n \log(\max\{n, m\})$, obviously, we only gain when the number of jobs n is large and the number of available speeds m small. Also, our constant factor K can be larger than K''.

Under a more detailed inspection, our algorithm is based on the fact that the input is made of $O(n)$ bounded integers, or equivalently, of $O(n)$ rational numbers with bounded numerators and denominators. This can be considered as a valid assumption because elementary operations needed in Algorithm 1 (it only uses additions and comparisons between inputs) only take a constant time under this assumption. The analysis of the arithmetic complexity in [LYY17] does not require that the job features are bounded integers. By taking into account the size of the input, the time complexity in [LYY17] will be of the form $K'' n \log(\max\{n, m\}) \log^2(B)$, where B is the maximal input size. Their algorithm is oblivious to the integrity of the input and both algorithms are oblivious to B. Obviously, our algorithm is only competitive over a restricted set of inputs (integer inputs with n large and B small).

We believe that the main contribution of our solution is twofold, on the one hand to show that computing the optimal speed schedule is not necessarily based on the critical interval, and on the other hand to show that this computation can be linear in the number of jobs to be scheduled.

As a side remark reinforcing this fact, there exist instances of jobs where Algorithm 2 cannot be used to find the critical interval. More precisely, by tuning the order in which the speeds are examined in line 23 of Algorithm 2, *all* the optimal speed schedules with integer switching times can be found by Algorithm 2 when the available speeds are consecutive. The following three facts are then true.

- One can find two sets of jobs with different critical intervals and different corresponding critical speeds for which there exists a common optimal speed schedule. This common solution can be the output of Algorithm 2 in both cases with a convenient choice of the order of the speeds in Line 23 of Algorithm 2. Consider the following example with a processor having two available speeds: $\{0, 1\}$. The first set is made of a single job $J_1 = (r_1, c_1, D_1) = (1, 3, 6)$. The second set is made of two jobs $J_2 = (1, 1, 6)$ and $J_3 = (2, 2, 5)$. The critical interval for the set $\{J_1\}$ is $I_1^c = [1, 6]$ with critical speed $s_1^c = 3/5$. In contrast, the critical interval for the set $\{J_2, J_3\}$ is $I_2^c = [2, 5]$ with critical speed $s_1^c = 2/3$. In both cases, if Algorithm 2 sweeps the speeds in increasing order in line 23, its solution is $(0, 0, 1, 1, 1)$.

- For any two sets of jobs with different critical intervals, $I_1^c \subset I_2^c$ (or/and different critical speeds $s_1^c < s_2^c$), there exists an optimal speed schedule for the first set that is not valid for the second set. Informally, this is true because the second set is more constrained and some "extreme" solution for the first set will not satisfy the more stringent constraints of the second set. In the previous example, the schedule $(1, 1, 0, 0, 1)$ is optimal for the set $\{J_1\}$ but it is not valid for the set $\{J_2, J_3\}$ because job J_3 is not completed before its deadline (the processor only executes one unit of work in the time interval $[2, 5]$ while job J_3 is of size 2 on the same interval).

- There exist examples where *some* optimal speed schedules cannot be found by an approach based on critical intervals. For example, using again the set $\{J_2, J_3\}$ with $J_2 = (1, 1, 6)$ and $J_3 = (2, 2, 5)$, the critical interval is $I_3^c = [2, 5]$ with critical speed $s_3^c = 2/3$. Once this critical interval is collapsed and the job J_3 that is included in I_3^c is removed, there remain the two intervals $[1, 2]$ and $[5, 6]$ and the job J_2. As a result, the new critical interval after collapsing becomes $I_4^c = [1, 3]$, with a critical speed $s_4^c = 1/2$. In this case, the optimal schedule $(0, 1, 1, 1, 0)$ can be found by Algorithm 2 but will never be discovered by approaches based on the critical interval, because all of them will use speed 0 exactly once in the critical interval I_3^c. This is illustrated in Fig.3.5.

3.5 Extensions

In this section, we show that Algorithm 2 can be adapted to compute an optimal solution in linear time even when switching from one speed to another has a time and/or energy cost, and when the power function is not convex.

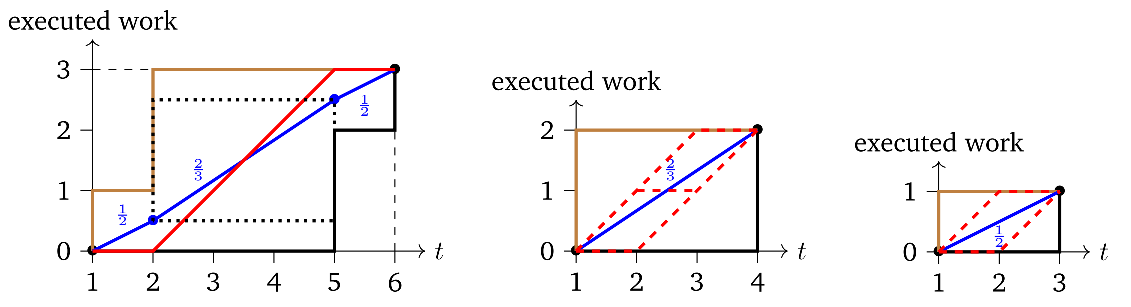


Figure 3.5.: A system made of two jobs $J_2 = (r_2, c_2, D_2) = (1, 1, 6)$ and $J_3 = (2, 2, 5)$ (left). The first critical interval is $I_3^c = [2, 5]$ (middle), which is materialized by the dotted rectangle in the left figure. Once I_3^c is collapsed, the second critical interval is $I_4^c = [1, 3]$ (right). The critical speed is $s^c = \frac{2}{3}$ on I_3^c and $s^c = \frac{1}{2}$ on I_4^c. All the optimal speed schedules obtained by critical interval methods are depicted as dashed red lines in the middle and right figures. The optimal speed schedule $(0, 1, 1, 1, 0)$ (red line in the left figure) cannot be found by algorithms based on critical intervals, but will be found by Algorithm 2.

3.5.1 Switching Time

So far, 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 the voltage of the circuit, while the time cost comes from the relocking of the Phase-Locked Loop when switching the frequency [Wu+05]. Burd and Brodersen have provided in [BB00] the equations to compute these two costs. In contrast with many DVFS studies (e.g., [BB00; BSA14; Li16; WRG16]), 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 ρ the time needed by the processor to change speeds. For the sake of simplicity we assume that the delay ρ is the same for each pair of speeds, but our formalization can accommodate different values of ρ, as computed in [BB00].

Consecutive Speeds

When there is a time delay, the executed work by the processor has *two* slope changes, at times τ_1 and τ_2, with $\tau_2 - \tau_1 = \rho$ (the red solid line in Fig. 3.6). We assume in this subsection that the speeds are consecutive. To take into account interpolations as in Section 3.4.3 with switching times inside integer intervals, we only have to modify the P_{ower} function with a penalty cost. We come back to this case in the end of the section to give a precise expression for this penalty cost.

Since $\rho \notin \mathbb{N}$, we cannot have both $\tau_1 \in \mathbb{N}$ and $\tau_2 \in \mathbb{N}$. As a consequence, one of the remaining work functions $w_{\tau_1}(.)$ or $w_{\tau_2}(.)$ will not be integer valued. This is not allowed by our approach.

The solution we propose is illustrated in Fig. 3.6. It consists in *shifting* the time τ_1 when the speed change is initiated so that the global behavior can be simulated by a single speed change that occurs at an integer time (t in Fig. 3.6). The *actual* behavior of the processor is represented by the red solid line, while the *simulated* behavior, which is equivalent in terms of the amount of

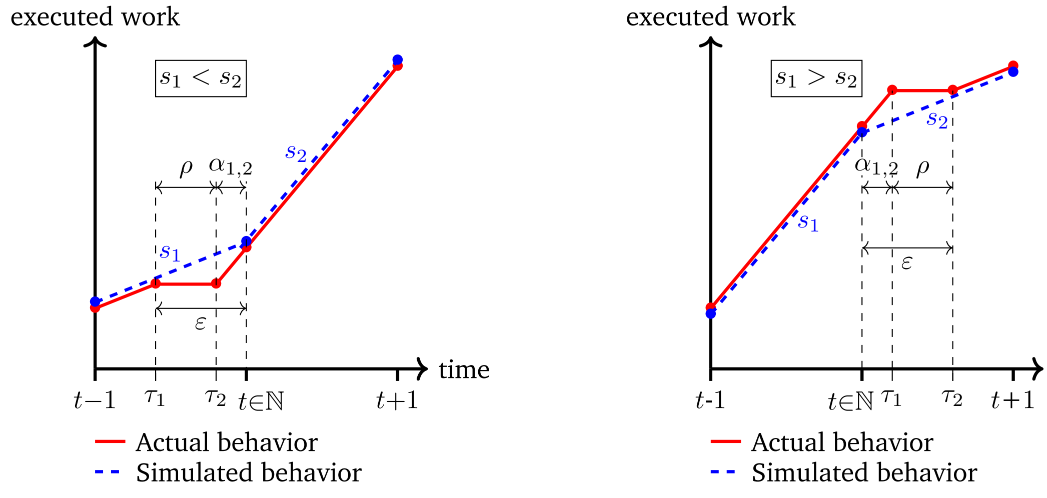


Figure 3.6.: Transformation of the time delay into an energy additional cost by shifting the switching point. The left figure corresponds to the $s_1 < s_2$ case and the right figure to the $s_1 > s_2$ case. The red line represents the actual behavior of the processor with a ρ time delay. The blue dashed line represents an equivalent behavior in terms of executed work, with no time delay.

work performed, is represented by the blue dashed line. The total amount of work done by the processor is identical in both cases at all integer times $t-1$, t, and $t+1$.

When $s_1 < s_2$, the speed change must be anticipated and occurs at $\tau_1 < t$ (Fig. 3.6-left). When $s_1 > s_2$, the speed change has to be delayed and occurs at $\tau_1 > t$ (Fig. 3.6-right). The exact computation of t_1 is similar in both cases and is straightforward.

One issue remains however, due to the fact that the consumed energy will not be identical between the real behavior and the simulated behavior. Indeed, it will be higher for the actual behavior for convexity reasons. This additional energy cost of the real processor behavior must therefore be added to the energy cost of the equivalent simulated behavior.

The value of ε and $\alpha_{1,2}$ as defined in Fig. 3.6, and the additional energy cost $h_\rho(s_1, s_2)$ incurred by this speed change are computed as follows. In the case $s_2 > s_1$, we have:

$$s_1\varepsilon = s_2\alpha_{1,2} = s_2(\varepsilon - \rho) \iff \varepsilon = \rho + \alpha_{1,2} = \frac{\rho s_2}{s_2 - s_1}. \tag{3.27}$$

During the time delay ρ, the energy is consumed by the processor as if the speed was s_1. The additional energy cost incurred in the actual behavior (the red solid line) compared with the simulated behavior (the blue dashed line), denoted $h_\rho(s_1, s_2)$, is therefore:

$$h_\rho(s_1, s_2) = \alpha_{1,2}(P_{ower}(s_2) - P_{ower}(s_1)). \tag{3.28}$$

Using the value of $\alpha_{1,2}$ from Eq. (3.27), this yields:

$$h_\rho(s_1, s_2) = \rho s_1 \left(\frac{P_{ower}(s_2) - P_{ower}(s_1)}{s_2 - s_1} \right). \tag{3.29}$$

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

$$h_\rho(s_1, s_2) = \rho s_2 \left(\frac{P_{ower}(s_1) - P_{ower}(s_2)}{s_1 - s_2} \right). \tag{3.30}$$

This additional energy due to speed changes can be taken into account in our model in the cost function by modifying the state space $\mathcal{W}$: the new state at time t becomes the *pair* (w_t, s_t): the remaining work at time t (*i.e.* w_t) and the speed that is used between t to $t+1$ (*i.e.* s_t).

We now wish to take into account both the energy and the time costs in Algorithm 2, which requires to modify the optimality equation of lines 25–26. At time t, recall that e is the executed work at time t, s is the speed used in the interval $[t-1, t]$ so that $e' = e - s$ was the state at time $t-1$, and s' is any previous speed used before time $t-1$. The new optimality equation must be changed to include the speed into the current configuration:

$$E_t^*(e, s) = \min_{s' \in \mathcal{S}} \left(P_{ower}(s) + h(s', s) + E_{t-1}^*(e - s, s') \right), \tag{3.31}$$

with the global switching cost $h(s', s) = h_e(s', s) + h_\rho(s', s)$, where $h_\rho(s', s)$ is given by Eq. (3.30) if $s' < s$ and by Eq. (3.29) if $s < s'$. When $s' = s$, $h(s', s) = 0$.

The rest of the algorithm is unchanged and the complexity remains linear.

Finally, if the speeds are not consecutive, we showed in Section 3.4.3 that speed changes can also happen inside an integer interval $[t, t+1]$ to emulate a non-available speed s by its two neighboring available speeds s_1 and s_2 by using interpolation. Taking these switching costs into account here is easier (no time shift is needed). One only needs to modify the optimality equation again. In Algorithm 2. This can be done at no additional complexity cost (each value for the speed smaller than $s_{\max}$ is still examined at most once for each point e) as follows.

Let $s \in \mathcal{S}$ be any available speed, and let $\underline{s}$ and $\overline{s}$ denote the closest available speeds in $\mathcal{S}$ from below and above respectively. The intermediate non-available speeds u strictly between $\underline{s}$ and s (resp. between s and $\overline{s}$) are such that $u = \alpha_u s + (1 - \alpha_u)\underline{s}$ (resp. $u = \alpha_u s + (1 - \alpha_u)\overline{s}$).

Again, the optimal energy in the configuration (e, s) at time t, $E_t^*(e, s)$, is the minimal energy consumption between time 0 and t if the current executed work is e and if the speed used just before time t is s. This can come from VDD-hoping emulating any speed u strictly between $\underline{s}$ and $\overline{s}$ as long as u is admissible (that is, $u \geq w_{t-1}^{e'}(1)$ and $u \leq M(t) - e'$). In this case, two speed changes occur: one between the previous speed s' and the first neighbor speed of u and one between the neighbor speed and s.

In this case, lines 25–26 in Algorithm 2 must be replaced by the following computation. The minimal principle applied to this new quantity says that $E_t^*(e, s)$ is the minimum between the two following terms:

$$\min_{\{s' \in \mathcal{S}, u \in \{\underline{s}+1 \ldots s\}\}} \left(\alpha_u P_{ower}(s) + (1 - \alpha_u) P_{ower}(\underline{s}) + h(s', \underline{s}) + h(\underline{s}, s) + E_{t-1}^*(e - u, s') \right), \tag{3.32}$$

and

$$\min_{\{s'\in\mathcal{S}, u\in\{s\ldots\overline{s}-1\}\}} \left(\alpha_u P_{ower}(s) + (1-\alpha_u)P_{ower}(\overline{s}) + h(s',\overline{s}) + h(\overline{s},s) + E^*_{t-1}(e-u,s')\right). \quad (3.33)$$

In the consecutive as well as the non-consecutive case, it is still true that there exists an optimal solution that only uses integer speeds (maybe through VDD-hoping in the non-consecutive case) in each time interval $[t, t+1]$, as long as the switching cost h satisfies a sub-additive inequality (For any s_1, s_2, s_3 in $\mathcal{S}$, $h(s_1, s_2) + h(s_2, s_3) \geq h(s_1, s_3)$), and are all negligible with respect to P_{ower} (s) for all $s \in \mathcal{S}$.

It is direct to check that if P_{ower} is convex, then this sub-additivity is true for the delay cost h_ρ. As for h_e (not detailed in this paper), the formulas provided in [BB00] also comply with the sub-additive inequality above.

Under sub-additivity and negligibility of h, each speed change will count, but will never compensate using the wrong speed: under these two assumptions, the optimal policy with switching costs will use each speed in $\mathcal{S}$ during the same total duration as the optimal policy without switching costs.

3.5.2 Non-Convex Power Function

In most real processors, measurements of the power function show that it is not a convex function of the speed. In most cases, a more realistic approximation is $P_{ower}(s) = P_{stat} + P_{dyn}(s)$, where $P_{dyn}(s)$ is convex. An even more accurate model is $P_{ower}(s) = P_{stat}(s) + P_{dyn}(s)$, where $P_{dyn}(s)$ is convex but the leakage power $P_{stat}(s)$ depends on s and is not convex.

If the power function is not convex, then it is well known that replacing $P_{ower}(.)$ by its convex hull $\widehat{P_{ower}}(.)$, and solving the speed selection problem with $\widehat{P_{ower}}(.)$ instead of $P_{ower}(.)$ also provides the optimal solution with $P_{ower}(.)$, by using speed replacements as described in Section 3.4.3.

Now we will present how to convexify the P_{ower} 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 $P_{ower}(.) : \mathcal{S} \to \mathbb{R}$.

If 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 3.7 depicts a non-convex power function P_{ower} 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 $g(s_1) < P_{ower}(s_1)$ (see Figure 3.7), 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 $P_{ower}(s_1)$ because of the convexity of function P_{ower}. This approach uses the V_{dd}-*hopping* technique.

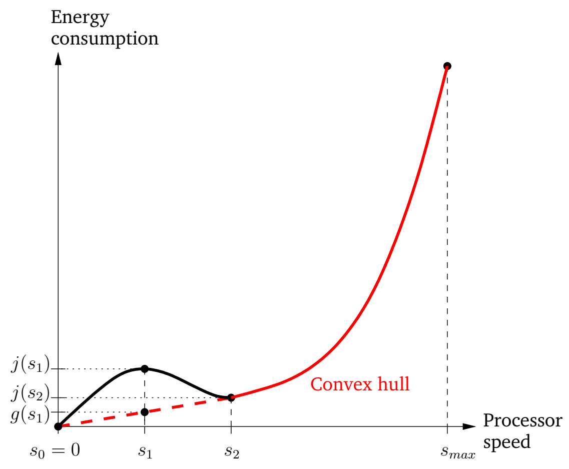


Figure 3.7.: Convexification of the power consumption function.

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 $P_{ower}(s) = Cs^{\alpha} + L(s)$, 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.

3.6 Conclusion

We have addressed the problem of minimizing off-line the total energy consumption required to execute a set of n real-time jobs on a single processor with varying speed. The goal is to find a sequence of processor speeds, chosen among a finite set of available speeds, such that no job misses its deadline and the energy consumption is minimal. Such a sequence is called an *optimal speed schedule*.

Our main result is that computing an optimal speed schedule can be done with a linear time complexity: Kn where n is the number of real-time jobs and K is a constant. This result holds for an arbitrary power function and may also take into account speed switching costs.

After the offline case studied in this chapter, that focuses on the situation where we know at any time all job features: arrival time, execution time and deadline, we examine the online case in the next chapter, *i.e.* the case where the processor discovers jobs while the processor is running. To begin we focus on the specific case of clairvoyant jobs. It means that at a certain time, all features of past and active jobs are known and future job features are only known statistically.

The following chapter will tackle the same problem, *i.e.* energy consumption minimization, and provides a solution that gives at each instant with these informations the optimal speed that the processor has to choose to minimize the expected energy consumption.

4 Online Minimization: Statistical Knowledge with *Clairvoyant* Jobs

After analysing the offline case, we use the same idea, *i.e.* dynamic programming, to solve the online case. Unlike the offline case, in the online case, we discover job characteristics, and especially the job arrival time only when they arrive. The first case of online situation on which we focus on in this thesis is the *Clairvoyant* case. As defined in Chapter 1, *Clairvoyant* Jobs means that at job arrivals, we know precisely its execution time, denoted c in the following, and also its relative deadline d. In other words, it means that at time t, one knows the past jobs, *i.e.* the jobs that are completed at t, and also entirely the active jobs, jobs that are in progress (by entirely it is job deadline and job execution time). Furthermore, there is a partial knowledge on the future jobs, *i.e.* jobs that arrive strictly after t: there is a statistical information. Thanks to these informations, an algorithm is build to minimize the energy consumption. Let us begin with the following section, Section 4.1, to present the existing literature on that topic, and also the previous algorithms, that have been designed in the past.

This chapter is based on [GGP19a], submitted to an international journal.

4.1 State of the Art

The starting point of this work is the seminal paper of Yao et al. [YDS95] followed by the paper of Bansal et al. [BKP07], both of which solve the following problem.

As presented in Chapter 2, let us consider $(r_i, c_i, d_i)_{i \in \mathbb{N}}$ 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 E is minimized. In this problem, E is the *dynamic* energy consumed by the processor: $E = \int_0^T P_{ower}(s(t))dt$, where T is the time horizon of the problem (in the finite horizon case) and $P_{ower}(s)$ is the power consumption when the speed is s.

This problem has been solved in Yao et al. [YDS95] when the power function P_{ower} is a *convex* function of the speed, in the *offline* case, *i.e.*, when *all* jobs are known in advance. Many

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

variants have been proposed to this offline solution, for different job and energy models (see *e.g.*, [Ayd+01]). However, in practice the *exact characteristics* of all the upcoming jobs cannot be known in advance, so the offline case is unrealistic.

Several solutions have been investigated by Bansal et al. in [BKP07] in the *online* 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 online algorithm introduced in [YDS95], 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 $P_{ower}(s) = s^\alpha$. 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 P_{ower} 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 this chapter 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 [Gru01; LS01; BS09] with a simpler framework, namely for a *single* job whose execution time is uncertain but whose release time and deadline are given, or in [PS01] by using heuristic schemes. Furthermore, [MCZ07] solves also an online 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 [MCZ07] 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 § 4.2.4). 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 chapter (§ 4.2), we present our job model and the problem addressed in the chapter. We define the state space of our problem (§ 4.2.3) and analyze its complexity (§ 4.2.4). In a second part (§ 4.3), 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 (§ 4.3.1), and a *Value Iteration* algorithm [Put05] for the infinite case (§ 4.3.2). Finally, we compute numerically the optimal policy in the finite and infinite horizon cases, and compare its performance with offline policies and "myopic" policies like (OA), oblivious to the arrival of future jobs (§ 4.4). 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 (§ 4.5). In appendix of this chapter, Appendix 4.7, we provide a proof that discrete speed changing and decision instants are optimal in this framework even when V_{dd}-*hopping* makes a continuous speed range available.

4.2 Presentation of the Problem

4.2.1 Job features, Processor Speed, and Power

The job model is the same as this one described in Chapter 2. Therefore all job J_i can be defined as a triplet (r_i, c_i, d_i).

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

In Chapter 3 we consider that we know all job features at any time, however in this chapter we do not know future job features, but 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 recall that the link between τ_i and r_i is done in Eq. (2.1).

In addition, we assume that the relative deadlines of the jobs are bounded by a maximal value, denoted Δ:

$$\Delta = \max_{i\in\mathcal{J}} \Delta_i \tag{4.1}$$

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,

$$\forall G \geq L,$$
$$\mathbb{P}(\tau_i = t, c_i = c, d_i = d | \tau_i \geq G) = \mathbb{P}(\tau_i = t, c_i = c, d_i = d | \tau_i \geq L). \tag{4.2}$$

We further define ℓ_t as the time elapsed between the last job arrival and t. As presented in Chapter 2, we assume the single processor can run at any time t at a speed $s(t)$ belonging to a finite set of integer speeds $\mathcal{S}$:

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

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 $s_1, \dots, s_k, s_{\max}$ 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 $P_{ower}(s(t))$, so that the energy consumption of the processor from time 0 to time T is computed as $E = \int_0^T P_{ower}(s(t))dt$. Usually, the power consumption P_{ower} is a convex increasing function of the speed (see [YDS95; BKP07]). This classical case is based on models of the power dissipation of CMOS circuits. Finer models use star-shaped functions [GNW05] to further take into account static leakage. In the present chapter, the function P_{ower} is *arbitrary*. However several structural properties of the optimal speed selection will only hold when the function P_{ower} is convex. In the numerical experiments (§ 4.4), several choices of P_{ower} 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 P_{ower} 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 § 4.5.

4.2.2 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 online, *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. Notice that in this model, unlike in [Gru01; LS01], the workload c_i and the deadline d_i are known at the release time of job i[2].

We now redefine the online energy minimization problem (MP) as:

> *Find online 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 speed changing instants (instants when the processor can change its speed) are also integers. We show in Appendix 4.7 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 (MP). 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 feasibility even when the speed of the processor changes

[2]When the actual workload can be smaller than WCET, our approach still applies by modifying the state evolution Eq. (4.4), to take into account early termination of jobs.

arbitrarily as proved in Appendix A.1, then (s^*, EDF) is also an optimal solution to problem (MP). 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 4.1. *The remaining work at time t is an increasing function $w_t(.)$ 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.

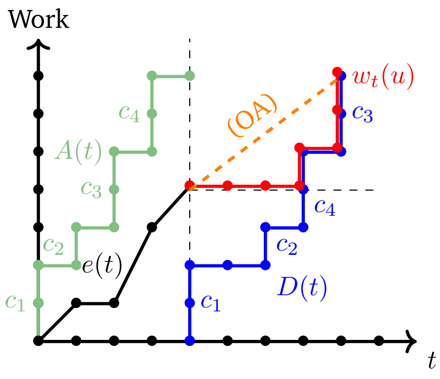


Figure 4.1.: Construction of the remaining work function $w_t(.)$ at $t = 5$, for jobs $J_1 = (1, 2, 4)$, $J_2 = (2, 1, 5)$, $J_3 = (3, 2, 6)$, $J_4 = (4, 2, 4)$, $J_5 = (5, 0, 0)$, and processor speeds $s_0 = 1, s_1 = 0, s_2 = 2, s_3 = 1$. $A(t)$ is the amount of work that has arrived before time t. $D(t)$ 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.

The definition of w_t is essential for the rest of the chapter. Let us illustrate it in Figure 4.1, which shows the set of jobs released just before $t = 5$, namely$J_1 = (1, 2, 4)$, $J_2 = (2, 1, 5)$, $J_3 = (3, 2, 6)$, $J_4 = (4, 2, 4)$, $J_5 = (5, 0, 0)$, 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 $t = 1+4 = 5$; the second step is for J_2 at $t = 2+5 = 7$; the third step is for J_4 at $t = 4+4 = 8$; the fourth step is for J_3 at $t = 3+6 = 9$. Hence the step for J_4 occurs *before* the step for J_3. This is because Figure 4.1 depicts the situation at $t = 5$. At $t = 4$ 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 4.1, 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. 4.1, we have depicted in red the staircase function $w_t(u)$ for $t = 5$.

Remark 4.1. *The online 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 4.1: the speed that* (OA) *would choose at time $t = 5$ is the slope of the orange 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 orange dotted line.*

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

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.

4.2.3 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 the set of the jobs released at t, denoted $\mathcal{E}_t$, as follows:

Definition 4.2. *The set $\mathcal{E}_t$ of the newly arrived jobs, released exactly at time t, is defined as:*

$$\mathcal{E}_t = \{J_i = (\tau_i, c_i, d_i), i \in \mathbb{N} \,|\, r_i = t\}. \tag{4.3}$$

where r_i is the release time of job J_i, defined in Eq. (2.1).

Furthermore, to take into account the deadlines of the new jobs, we define in Def. 4.3 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 4.3. *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 4.1. *Let s_{t-1} be the processor speed at time $t-1$. Then at time t the remaining work function becomes:*

$$w_t(.) = \mathbb{T}\left[(w_{t-1}(.) - s_{t-1})^+\right] + a_t(.) \tag{4.4}$$

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

$$\ell_t = \begin{cases} 0 & \text{if } \mathcal{E}_{t-1} \neq \emptyset \\ (\ell_{t-1}+1) \wedge L & \text{otherwise.} \end{cases} \tag{4.5}$$

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 $(w_{t-1}(.) - s_{t-1})^+$. 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 $a_t(.)$.

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. 4.2 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}.

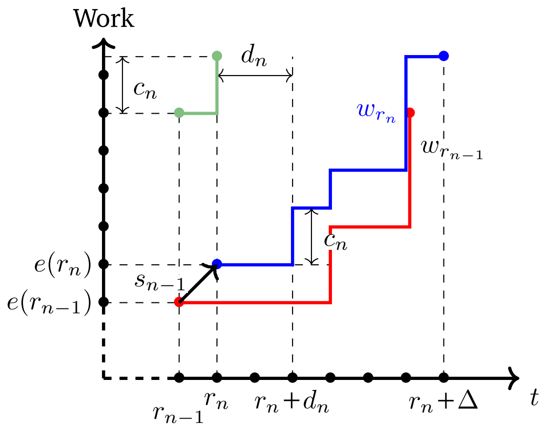


Figure 4.2.: State change following a job arrival at time r_n. The red line corresponds to the previous remaining work function. The blue line corresponds to the new remaining work function.

4.2.4 Size of $\mathcal{W}$

Feasible Policies

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 schedulability of the set 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:

$$M \leq 2s_{\max}\Delta \tag{4.6}$$

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.

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

Proposition 4.1. *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:*

$$Q(\Delta) \leq \binom{\Delta s_{\max}+\Delta-1}{\Delta-1} \tag{4.7}$$

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

Proof. A state is an increasing integer functions $w_t(.)$. 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:

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

□

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. 4.2.

Proposition 4.2. *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:*

$$Q(\Delta, C) = \sum_{y_1=0}^{C} \sum_{y_2=y_1}^{2C} \sum_{y_3=y_2}^{3C} \dots \sum_{y_\Delta = y_{\Delta-1}}^{\Delta C} 1 \tag{4.8}$$

It can be computed in closed form as:

$$\begin{aligned} Q(\Delta, C) &= \frac{1}{1 + C(\Delta+1)} \binom{(C+1)(\Delta+1)}{\Delta+1} && (4.9) \\ &\approx \frac{e}{\sqrt{2\pi}} \frac{1}{(\Delta+1)^{3/2}} (e\,C)^{\Delta} && (4.10) \end{aligned}$$

Proof. The proof is postponed to Appendix A.2. □

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

4.3 Markov Decision Process (MDP) Solution

We denote by $\boldsymbol{x} = (w(.), \ell)$ a *state* of the MDP, defined below. It is composed by a remaining work function denoted $w(.)$, and the time elapsed since the latest job arrival denoted ℓ. One has to remark that the state is different as in Chapter 3. We denote by $\mathcal{X}$ the state space (the set of all possible states).

As explained in § 4.2.4, 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 $\boldsymbol{x}$ 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 offline 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 $\boldsymbol{x}_t = (w_t, \ell_t)$, that is $s(t) := \sigma_t^*(\boldsymbol{x}_t)$.

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 $\boldsymbol{x}$ and at time t, the distribution $(\phi_{\boldsymbol{x}})_{\boldsymbol{x}\in\mathcal{X}}$ of the work that arrives at t. For any $(c_1, \dots, c_\Delta)$:

$$\phi_{\boldsymbol{x}}(t, c_1, \dots, c_\Delta) = \mathbb{P}\left(a_t = \sum_{k=1}^{\Delta} c_k H_{k+t} \,|\, \boldsymbol{x}_t = \boldsymbol{x}\right) \tag{4.11}$$

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

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

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.

4.3.1 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 $\boldsymbol{x}_0$, then

$$E^*(\boldsymbol{x}_0) = \min_{\sigma}\left(\mathbb{E}\left(\sum_{t=0}^{T} P_{ower}(\sigma_t(\boldsymbol{x}_t))\right)\right) \tag{4.12}$$

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: $\sigma_t(\boldsymbol{x})$ is the speed used at time t if the state is $\boldsymbol{x}$. The only constraint on $\sigma_t(\boldsymbol{x})$ is that it must belong to the set of available speeds, *i.e.*, $\sigma_t(\boldsymbol{x}) \in \mathcal{S}$, and it must be large enough to execute the remaining work at the next time step:

$$\forall t, \sigma_t(\boldsymbol{x}) \geq w(1) \tag{4.13}$$

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

$$\mathcal{A}(\boldsymbol{x}) = \left\{ s \in \mathcal{S} \text{ s.t. } s \geq w(1) \right\} \tag{4.14}$$

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

Dynamic Programming Algorithm (DP)

We use a backward induction (Dynamic Programming) to recursively compute the expected energy consumption E^* and the optimal speed policy σ^*. We use the Backward Induction Algorithm from [Put05] (p. 92). We obtain an *optimal policy* that gives the processor speed that one must apply in order to minimize the energy consumption (Algorithm 3).

Algorithm 3: Dynamic Programming Algorithm (DP) to compute the optimal speed for each state and each time.

```
$t \leftarrow T$                                                   % time horizon
for all $\boldsymbol{x} \in \mathcal{X}$ do $E_t^*(\boldsymbol{x}) \leftarrow 0$ end for
while $t \geq 1$ do
  for all $\boldsymbol{x} \in \mathcal{X}$ do
    $E_{t-1}^*(\boldsymbol{x}) \leftarrow \min_{s \in \mathcal{A}(\boldsymbol{x})} \left( P_{ower}(s) + \sum_{\boldsymbol{x}' \in \mathcal{W}} P_t(\boldsymbol{x}, s, \boldsymbol{x}') E_t^*(\boldsymbol{x}') \right)$
    $\sigma_{t-1}^*[\boldsymbol{x}] \leftarrow \operatorname{argmin}_{s \in \mathcal{A}(\boldsymbol{x})} \left( P_{ower}(s) + \sum_{\boldsymbol{x}' \in \mathcal{W}} P_t(\boldsymbol{x}, s, \boldsymbol{x}') E_t^*(\boldsymbol{x}') \right)$
  end for
  $t \leftarrow t - 1$                                             % backward computation
end while
return all tables $\sigma_t^*[.]$  $\forall t = 0 \ldots T-1$.
```

The complexity to compute the optimal policy $\sigma_t^*(\boldsymbol{x})$ 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 *offline*. At runtime, the processor simply considers the current state $\boldsymbol{x}_t$ at time t and uses the pre-computed speed $\sigma_t^*(\boldsymbol{x}_t)$ to execute the job with the earliest deadline.

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

At runtime, the processor computes the current state $\boldsymbol{x}_t$ and simply uses a Table Look-Up algorithm (TLU) to obtain its optimal speed $\sigma_t^*[\boldsymbol{x}_t]$, the speed tables having been computed offline by (DP).

This algorithm, called $(\mathrm{TLU_{DP}})$, is shown in Algorithm 4. The size of the table is $T \times Q(\Delta)$ and the runtime cost for $(\mathrm{TLU_{DP}})$ is $\mathcal{O}(1)$.

Algorithm 4: Runtime process $(\mathrm{TLU_{DP}})$ used by the processor to apply the optimal speed

```
For Each $t = 0 \ldots T-1$
    Update $\boldsymbol{x}_t$ using Eq. (4.4)
    Set $s := \sigma_t^*[\boldsymbol{x}_t]$
    Execute the job(s) with earliest deadline at speed $s$ for one time unit
End
```

4.3.2 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

$$g^* = \min_{\sigma} \mathbb{E}\left(\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} P_{ower}(\sigma(\boldsymbol{x}_t)) \right) \tag{4.15}$$

under the constraint that no job misses its deadline.

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 [Put05] 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. (4.15) always exists (see [Put05]) and can be computed with an arbitrary precision using a *value iteration* algorithm (VI), presented in the next section.

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 5: 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 $\boldsymbol{x} \in \mathcal{W}$ do
    $u^{n+1}(\boldsymbol{x}) \leftarrow \min_{s \in \mathcal{A}(\boldsymbol{x})} \left( P_{ower}(s) + \sum_{\boldsymbol{x}' \in \mathcal{W}} P(\boldsymbol{x}, s, \boldsymbol{x}') u^n(\boldsymbol{x}') \right)$
  end for
  $n \leftarrow n + 1$
end while
Choose any $\boldsymbol{x} \in \mathcal{W}$ and let $g^* \leftarrow u^n(\boldsymbol{x}) - u^{n-1}(\boldsymbol{x})$
for all $\boldsymbol{x} \in \mathcal{W}$ do
  $\sigma^*[\boldsymbol{x}] \leftarrow \operatorname{argmin}_{s \in \mathcal{A}(\boldsymbol{x})} \left( P_{ower}(s) + \sum_{\boldsymbol{x}' \in \mathcal{W}} P(\boldsymbol{x}, s, \boldsymbol{x}') u^n(\boldsymbol{x}') \right)$
end for
return $\sigma^*$
```

In Algorithm 5, 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 5 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 ($\sigma^*[\boldsymbol{x}]$).

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 § 4.4).

Runtime Process: Table Look-Up $(\mathrm{TLU_{VI}})$

As for $(\mathrm{TLU_{DP}})$, at each integer time $t \in \mathbb{N}$, the processor computes its current state $\boldsymbol{x}_t$ and retrieves its optimal speed $s := \sigma^*[\boldsymbol{x}]$ by looking-up in the table σ^* that was pre-computed by (VI). This algorithm is identical to Algorithm 4, except for the the size of the table, which is $Q(\Delta)$.

4.3.3 Feasibility Issues

Let us recall that, according to Definition 2.2, a policy 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.

The proof of the feasibility of (DP) and (VI) when the speed decision times are integer numbers has been done in Section 8.8 of Chapter 8, a specific chapter entirely devoted to feasibility analysis.

As a final remark, not all online 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 § 4.3.5, requires additional constraints on a schedulable set of jobs to guarantee feasibility (see Section 8.5 in Chapter 8).

4.3.4 Bounded Job Sizes

As in § 4.2.4, 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:

$$s_{\max} \geq C \tag{4.16}$$

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 $a_t(.)$ is the work quantity arrived at time t (see Def. 4.3).

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 4.3. *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 $s_{\max} \geq C$.*

4.3.5 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.

Comparison with Optimal Available (OA)

Optimal Available (OA) is an online speed policy introduced in [YDS95], which chooses the speed $s^{(\mathrm{OA})}(\boldsymbol{x}_t)$ at time t and in state $\boldsymbol{x}_t$ 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 $\boldsymbol{x}_t$, the (OA) policy uses the speed

$$s^{(\mathrm{OA})}(\boldsymbol{x}_t) = \max_u \frac{w_t(u)}{u} \tag{4.17}$$

where $w_t(.)$ is the remaining work function computed by Eq. (4.4).

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

Proposition 4.4. *Both in the finite or infinite case, the optimal speed policy σ^* satisfies*

$$\sigma^*(\boldsymbol{x}) \geq s^{(\mathrm{OA})}(\boldsymbol{x}) \tag{4.18}$$

for all state $\boldsymbol{x} \in \mathcal{X}$, 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 $s^{(\mathrm{OA})}(\boldsymbol{x}_t) = \max_u \frac{w_t(u)}{u}$ is reached for $u = 1$ (*i.e.*, $s^{(\mathrm{OA})}(\boldsymbol{x}_t) = w_t(1)$), then $\sigma^*(\boldsymbol{x}_t) \geq s^{(\mathrm{OA})}(\boldsymbol{x}_t)$ by definition because the set of admissible speeds $\mathcal{A}(\boldsymbol{x}_t)$ only contains speeds higher than $w_t(1)$ (see Eq. (4.17)).

If the maximum is reached for $u > 1$, then $\mathcal{A}(\boldsymbol{x}_t)$ may enable the use of speeds below $w_t(1)$.

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

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

Now, if the set of speeds is finite, then the optimal value of $\sigma^*(\boldsymbol{x}_t)$ must be one of the two available speeds in $\mathcal{S}$ surrounding $\sigma^{(\mathrm{OA})}(\boldsymbol{x}_t)$. Let s_1 and s_2 in $\mathcal{S}$ be these two speeds, *i.e.*, $s_1 < \sigma^{(\mathrm{OA})}(\boldsymbol{x}_t) \leq s_2$, and assume again that the max in Eq. (4.17) is not reached for $t = 1$. If the smallest speed s_1 is chosen as the optimal speed, this implies that further choices for $\sigma^*(\boldsymbol{x}_{t+i})$ will have to be greater or equal to s_2, to compensate for the work surplus resulting from choosing a speed below $\sigma^*(\boldsymbol{x}_t)$. This implies that it is never sub-optimal to choose s_2 in the first place (by convexity of the P_{ower} function).

This trajectory based argument is true almost surely, so that the inequality $\sigma^*(\boldsymbol{x}_t) \geq s^{(\mathrm{OA})}(\boldsymbol{x}_t)$ will also hold for the *expected* energy over both a finite or infinite time horizon. □

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 $\boldsymbol{x}$.

[4]Actually, we use the fact that the sum $\sum_{i=0}^{u-1} j(s)$ is Schur-convex when j is convex (see [MO79]).

Definition 4.4. *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:*

$$\begin{aligned} \forall \quad \gamma, \delta, \quad & \mathbb{P}(c_2 \geq \gamma, d_2 \leq \delta) \leq \mathbb{P}(c_1 \geq \gamma, d_1 \leq \delta) \\ \forall \quad t \in \mathbb{N}, \quad & \mathbb{P}(\tau_1 = t) = \mathbb{P}(\tau_2 = t). \end{aligned} \tag{4.19}$$

Moreover, by denoting $(i^1_1, \ldots, i^1_\Delta)$ the work quantity that arrives at time t for $(\mathrm{MP})_1$, and $(i^2_1, \ldots, i^2_\Delta)$ the work quantity that arrives at time t for $(\mathrm{MP})_2$, we have:

$$\begin{aligned} & \forall t, i^1_1, \ldots i^1_\Delta, i^2_1, \ldots i^2_\Delta, \\ & \mathbb{P}(t, w_{t+1} \geq i^1_1, \ldots, w_{t+\Delta} \geq i^\Delta_\Delta) \leq \mathbb{P}(t, w_{t+1} \geq i^2_1, \ldots, w_{t+\Delta} \geq i^2_\Delta) \end{aligned}$$

Proposition 4.5. *If $\mathcal{T}_2 \leq_s \mathcal{T}_1$, then:*

1. *over a finite time horizon T, the total energy consumption satisfies $E^{(2)} \leq E^{(1)}$ (computed with Eq. (4.12));*

2. *in the infinite time horizon case, the average energy consumption per time unit satisfies $g^{(2)} \leq g^{(1)}$ (computed with Eq. (4.15)).*

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 $t \leq T$, $J^{(1)}_t = (\tau^1_t, c^1_t, d^1_t)$ and $J^{(2)}_t = (\tau^2_t, c^2_t, d^2_t)$ with $t = \tau^1_t = \tau^2_t$, $c^2_t \leq c^1_t$ and $d^2_t \geq d^1_t$ (see [MS02]). 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 $E^{(2)} \leq E^{(1)}$.

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 $\mathcal{T}_2$. □

4.4 Numerical Experiments

4.4.1 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 online 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 § 4.4.3 and § 4.4.3, 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.

4.4.2 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 3 and 5 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. (4.8) to traverse the state space.

The implementation of Algorithms 3 and 5 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.

4.4.3 Experimental Set-up, Finite Case

Our experiments are done in two steps:

- Firstly, we compute the optimal speeds for each possible state $\boldsymbol{x} \in \mathcal{X}$. For this, we use Algorithm 3 (DP) or 5 (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 § 4.4.1) over which we use our $(\mathrm{TLU}_{\mathrm{DP}})$ solution or other solutions (*e.g.*, offline 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 *offline* solution (see § 4.4.3). 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 § 4.4.3). In the periodic case, where jobs are more predictable, the gap with (OA) grows to about 50% (see § 4.4.3).

Comparison with the Offline Solution

To evaluate our online algorithm, we compare it with the offline solution computed on a simulated set of jobs, characteristics of which are described in Example 4.1. We draw the aggregated work done by the processor (the respective speeds are the slopes) in two cases:

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

Example 4.1. *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 $P_{ower}(s) = s^3$.*

A job of size 0 at some time instant t is the same as no job at all at time t. In Example 4.1, 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 4.1, the maximal speed is large enough so that schedulability is not an issue: $5 = s_{\max} > C = 2$ (§4.3.3). Note that, in contrast with $(\mathrm{TLU}_{\mathrm{DP}})$, some jobs created by task $\boldsymbol{T}_1$ might not be schedulable under (OA).

The result over one typical simulation of run for Example 4.1 is displayed in Figure 4.3. As expected, $(\mathrm{TLU}_{\mathrm{DP}})$ consumes more energy than the offline case. The differences in the chosen speeds are the following: (i) speed 0 is used once by $(\mathrm{TLU}_{\mathrm{DP}})$ but is never used by the offline solution; (ii) speed 2 is used 5 times by $(\mathrm{TLU}_{\mathrm{DP}})$ and only 4 times in the offline 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 offline solution is $46\ J$, while the total energy consumption under the $(\mathrm{TLU}_{\mathrm{DP}})$ solution is $52\ J$, the difference being 13% of the total energy consumption.

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

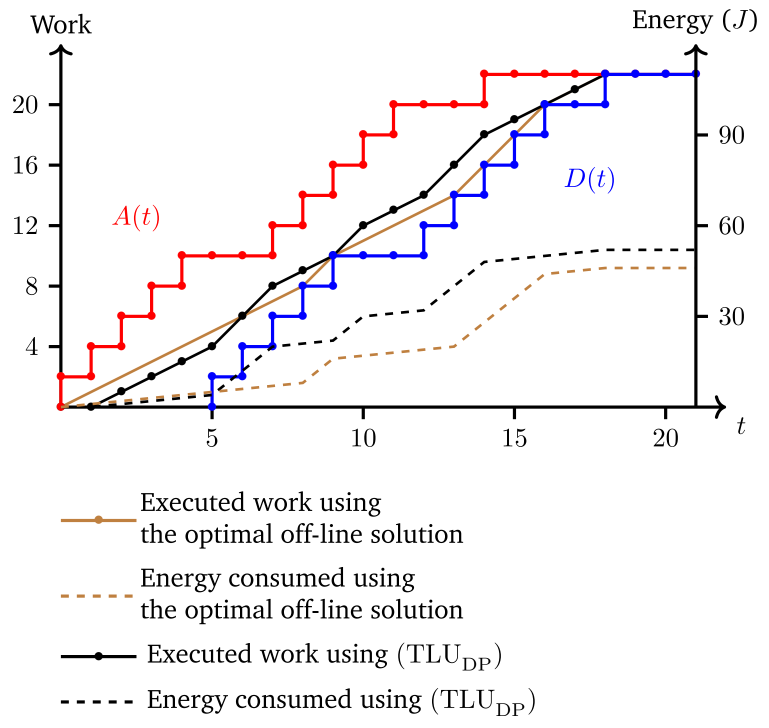


Figure 4.3.: Comparison of the executed work of offline and $(\mathrm{TLU_{DP}})$ solution on one simulation of Example 4.1. As defined before, $A(t)$, the red curve, is the workload arrived between 0 and T, and $D(t)$, the blue curve, is the workload deadlines from 0 to T; Brown curve: work executed using the optimal offline speeds; Black curve: work executed using the speed selection computed by $(\mathrm{TLU_{DP}})$.

Comparison with (OA), Sporadic Tasks

Recall that, under (OA), the processor speed at time t in state $\boldsymbol{x}_t$ 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 $s^{(\mathrm{OA})}(\boldsymbol{x}_t)$ 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 4.2, which consists of two sporadic tasks, using the same modelling technique as in Example 4.1 by fixing $c_2 = \{0, 3, 6\}$.

Example 4.2. *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 $P_{ower}(s) = s^3$.*

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 $(\mathrm{TLU}_{\mathrm{DP}})$ over (OA) in percentage. The gain percentage of $(\mathrm{TLU}_{\mathrm{DP}})$ 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 $(\mathrm{TLU}_{\mathrm{DP}})$ online takes a constant time to select the speed (table look-up) while using (OA) online takes $\mathcal{O}(\Delta)$ to compute the value $\max_u \frac{w_t(u)}{u}$.

Figure 4.4 shows a comparison between (OA) and $(\mathrm{TLU}_{\mathrm{DP}})$. The total work executed by the $(\mathrm{TLU}_{\mathrm{DP}})$ solution is always above the total work executed by (OA), as stated in Proposition 4.4. Moreover, the consumed energy is more important at the beginning with $(\mathrm{TLU}_{\mathrm{DP}})$ than with (OA), because we anticipate the work that will arrive in the future. The processor executes more work so it consumes more energy with $(\mathrm{TLU}_{\mathrm{DP}})$ before time $t = 11$; but after this time, it's the opposite, the energy consumed by $(\mathrm{TLU}_{\mathrm{DP}})$ is lower than the energy consumed by (OA). Over the whole period, $(\mathrm{TLU}_{\mathrm{DP}})$ outperforms (OA): The total energy consumption for (OA) is $711\ J$ (dashed brown curve) while that for $(\mathrm{TLU}_{\mathrm{DP}})$ is $639\ J$ (dashed black curve). As a result, $(\mathrm{TLU}_{\mathrm{DP}})$ 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 offline.

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 4.3 (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

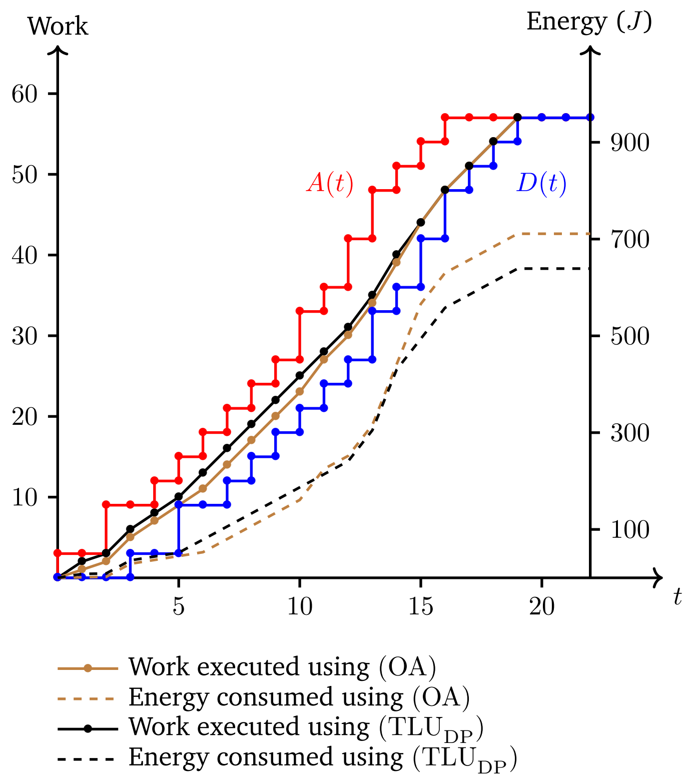


Figure 4.4.: Comparison of the executed work between (OA) and $(\mathrm{TLU_{DP}})$ solutions, with fixed deadlines $d_n = 3$, size $c_n = \{0, 3, 6\}$ w.r.p. $\{0.2, 0.6, 0.2\}$, and processor speeds in $\{0, 1, 2, 3, 4\}$. As defined before, $A(t)$ (red curve) is the workload arrived between 0 and T, while $D(t)$ (blue curve) is the workload deadlines from 0 to T.

equal to zero. On the other examples (Examples 4.4 and 4.5), we can perform the same analysis as above to show that the probability matrix depends on the time.

This Section displays three examples, Examples 4.3, 4.4, and 4.5, which consider different cases where we have several periodic tasks that have not necessarily the same offset and the same periodicity.

Example 4.3. *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 $(\mathrm{TLU}_{\mathrm{DP}})$, and 825 J for (OA), so more than 60% bigger. Here $(\mathrm{TLU}_{\mathrm{DP}})$ has a clear advantage because the job characteristics are highly predictable.

Example 4.4. *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 4.4, the energy consumed by $(\mathrm{TLU}_{\mathrm{DP}})$ 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 4.5. *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 $\boldsymbol{T}_1, \dots, \boldsymbol{T}_3$ and $\boldsymbol{T}_5, \dots, \boldsymbol{T}_7$ 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 4.5, the energy consumed by $(\mathrm{TLU}_{\mathrm{DP}})$ 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 4.3 to 4.5 are $T = 20$, $\mathcal{S} = \{0, 1, 2, 3, 4, 5\}$ and $P_{ower}(s) = s^3$.

Table 4.1 summarizes these results.

Table 4.1.: Comparisons between (OA) and $(\mathrm{TLU}_{\mathrm{DP}})$.

example	gain over (OA)	95% confidence interval
Ex. 4.3 (2 tasks)	56.44%	$[56.21, 56.68]$
Ex. 4.4 (4 tasks)	29.04%	$[28.84, 29.24]$
Ex. 4.5 (7 tasks)	46.88%	$[46.71, 47.04]$

In all these cases, $(\mathrm{TLU}_{\mathrm{DP}})$ 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 $(\mathrm{TLU}_{\mathrm{DP}})$ is based is more useful here than in the sporadic case.

4.4.4 Numerical Experiments, Infinite Case

In this section, we run algorithm (VI) (Algorithm 5) to compute the optimal speed to be used at each time step over an infinite horizon. We fix the stopping criterion in Algorithm 5 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 4.6. *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 4.7. *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 $P_{ower}(s) = s^2$. The only difference between Examples 4.6 and 4.7 are the deadlines.

The results of our computations are displayed in Figure 4.5. 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 4.6 and 4.7, together with the theoretical lower bound.

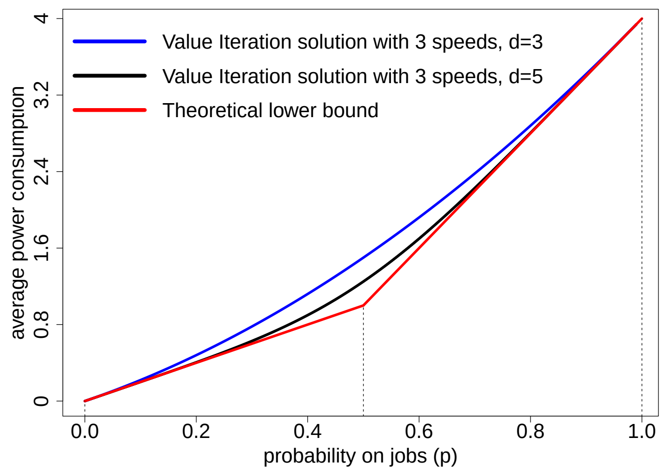


Figure 4.5.: Average energy consumption per time unit for $(\mathrm{TLU}_{\mathrm{VI}})$: theoretical lower bound (red curve), deadlines equal to 3 (black curve, Example 4.6), and deadline equal to 5 (blue curve, Example 4.7).

The different curves in Figure 4.5 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 5) 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 4.5, the higher the arrival rate, the higher the average energy consumption: both curves are increasing.

Proposition 4.5 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 $p \in [0, 0.20] \cup [0.80, 1]$.

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 $P_{ower}(s) = s^2$), 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:

$$g^\infty = 2p \times 1^2 + (1 - 2p) \times 0^2 = 2p \tag{4.20}$$

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:

$$g^\infty = (2p - 1) \times 2^2 + (2 - 2p) \times 1^2 = 6p - 2 \tag{4.21}$$

By combining Eqs. (4.20) and (4.21), we obtain the lower bound on g:

$$g^{\infty}(p) = \begin{cases} 2p & \text{if} \quad p \leq 1/2 \\ 6p - 2 & \text{if} \quad p \geq 1/2 \end{cases} \tag{4.22}$$

This is the red curve in Figure 4.5.

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 4.6, is computed as follows:

$$\frac{\texttt{Energy}_{(\mathrm{VI})} - \texttt{Energy}_{(\mathrm{DP})}}{\texttt{Energy}_{(\mathrm{DP})}} \tag{4.23}$$

This fraction computes the relative difference between the infinite horizon case algorithm (Algorithm 5) and the the finite horizon case algorithm (Algorithm 3). Besides, the cost of (OA) versus $(\mathrm{TLU}_{\mathrm{DP}})$, also represented in Figure 4.6 as a dashed red curve, is defined as follows:

$$\frac{\texttt{Energy}_{(\mathrm{OA})} - \texttt{Energy}_{(\mathrm{DP})}}{\texttt{Energy}_{(\mathrm{DP})}} \tag{4.24}$$

Computations were done on Example 2 with $10,000$ simulations. They are summarized in Table 4.2.

Table 4.2.: 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%	$6.19.10^{-5}$%	
(OA) vs (DP)	4.3%	4.3%	4.2%	

One can notice in Table 4.2 as well as on the blue curve in Figure 4.6 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 4.6 confirm the results obtained in Example 4.2 before, because whatever the considered time horizon, the gain of (DP) in comparison with (OA) ranges between 4% and 5.5%.

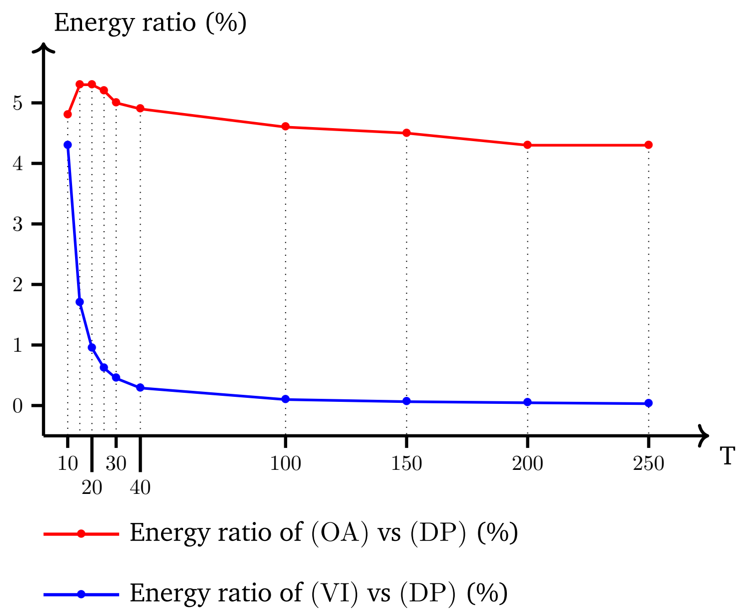


Figure 4.6.: Influence of the time horizon T on the energy difference between (OA) and (DP), and between (VI) and (DP), with Example 4.2.

4.5 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.

4.5.1 Convexification of the Power Consumption Function

Our general approach does not make any assumption on the power function $P_{ower}(.)$. Our algorithms (DP) and (VI) will compute the optimal speed selection for any function $P_{ower}(.)$. However structural properties (including the comparison with (OA) and the monotonicity) require the convexity assumption. It is therefore desirable to convexify the power function. This can be done as presented in Section 3.5.2 of Chapter 3. We replace $P_{ower}(.)$ by its convex hull $\widehat{P_{ower}}(.)$, and solve the speed selection problem with $\widehat{P_{ower}}(.)$ instead of $P_{ower}(.)$. It also provides the optimal solution with $P_{ower}(.)$, by using speed replacements as described in Section 3.4.3.

4.5.2 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, as explained and presented in Chapter 3, in all synchronous CMOS circuits, changing speeds does consume time and energy. One advantage of our formulation is that it can accommodate to arbitrary energy cost to switch from speed s to s'. As in Chapter 3, in the sequel, we denote this energy cost by $h_e(s, s')$.

We recall that the time needed by the processor to change speeds is noted ρ. For the sake of simplicity we assume that the delay ρ is the same for each pair of frequencies, but our formalization can accommodate different values of ρ, as computed in [BB00]. 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 τ_1 and τ_2, with $\tau_2 - \tau_1 = \rho$ (see the red solid line in Figure 3.6). The problem is that, since in general $\rho \notin \mathbb{N}$, we cannot have both $\tau_1 \in \mathbb{N}$ and $\tau_2 \in \mathbb{N}$. As a consequence, one of the remaining work functions w_{τ_1} or w_{τ_2} of the state states $\boldsymbol{x}_{\tau_1}$ or $\boldsymbol{x}_{\tau_2}$ 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 3.6) by a *simulated* behavior, equivalent in terms of the amount of work performed (represented by the blue dashed line in Figure 3.6). 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 $t-1$, t, and $t+1$ in Figure 3.6). 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* τ_1 such that $t \in \mathbb{N}$.

When $s_1 < s_2$, the speed change must be anticipated and occurs at $\tau_1 < t$ (left figure). When $s_1 > s_2$, the speed change has to be delayed and occurs at $\tau_1 > t$ (right figure). The exact computation of τ_1 is similar in both cases and is straightforward.

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 τ_1, the processor needs to be "clairvoyant", *i.e.*, it needs to know in advance (before τ_1) the characteristics of the job arriving at time t. 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 in the two cases (real and simulated) is identical.

In Chapter 3, the two additional costs (case $s' < s$ and $s' > s$) link to the time delay, $h_\rho(s', s)$ and $h_\rho(s', s)$, are computed.

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 $(\boldsymbol{x}_t, s_{t-1})$.

Taking into account both the energy cost and the time cost, the new main step of the (DP) Algorithm 3 becomes:

$$E^*_{t-1}(\boldsymbol{x}, s) \leftarrow \min_{s' \in \mathcal{A}(\boldsymbol{x})} \Big(P_{ower}(s) \quad + \quad h_\rho(s, s') + h_e(s, s') \\ + \sum_{\boldsymbol{x}' \in \mathcal{W}} P_t(\boldsymbol{x}, s', \boldsymbol{x}') E^*_t(\boldsymbol{x}', s') \Big) \tag{4.25}$$

with $h_\rho(s, s') = 0$ when $s = s'$, and otherwise given by Eq. (3.30) if $s' < s$ and Eq. (3.29) if $s < s'$. The rest of the analysis is unchanged.

4.5.3 Taking into Account the Cost of Context Switches

In the core of this chapter, 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 [BBA10; Bra11; BG16]. Nevertheless, in the following, we present a solution where we take into account this context switch delay.

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.

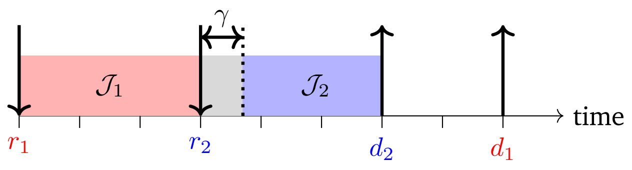


Figure 4.7.: Impact of a context switch on the execution time.

Figure 4.7 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.

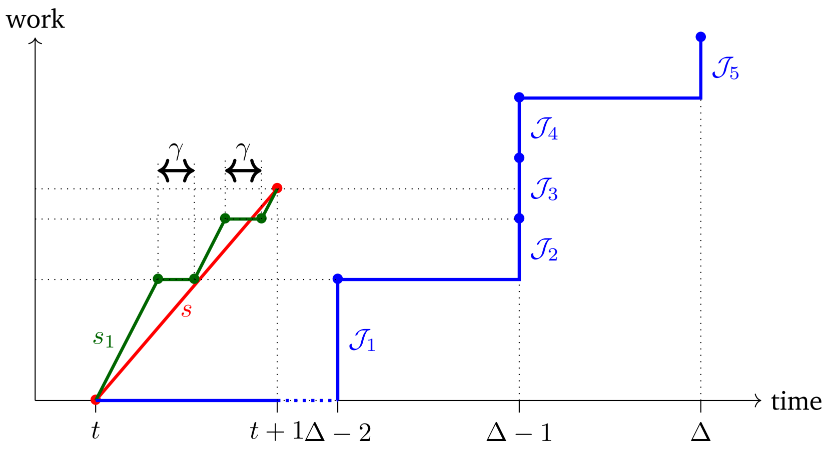


Figure 4.8.: Compensation of the impact of the context switches on the executed work by using a higher speed. The released jobs are $\mathcal{J}_i = (r_i, c_i, d_i)$, for $1 \leq i \leq 5$, with $r_5 + d_5 = \Delta$, $r_4 + d_4 = r_3 + d_3 = r_2 + d_2 = \Delta - 1$, and $r_1 + d_1 = \Delta - 2$.

Figure 4.8 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 4.5.2, 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 $w(i), i \in \{0, ..., \Delta\}$, 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:

$$\overline{\mathbf{x}}_t = (\overline{w}_t, \ell_t) \tag{4.26}$$

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

$$\overline{w}_t = \left[(\rho_1^1, ..., \rho_1^{k_1}), ..., \underbrace{(\rho_i^1, ..., \rho_i^{k_i})}_{\substack{\text{remaining work quantity of jobs} \\ \text{with absolute deadline } t+i}}, ..., (\rho_\Delta^1, ..., \rho_\Delta^{k_\Delta}) \right] \tag{4.27}$$

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. (2.1)). 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 $\overline{\boldsymbol{x}}_{t+1}$ becomes $(\overline{w}_{t+1}, \ell_{t+1})$, where $\overline{w}_{t+1}$ is:

$$\begin{aligned} \overline{w}_{t+1} = \Bigg[& \Big((\rho_2^1 - f(1,1))^+, ..., (\rho_2^{k_1} - f(1, k-1))^+ \Big), ..., \\ & \Big((\rho_{d_n}^1 - f(d_n, 1))^+, ..., (\rho_{d_n}^{k_{d_n}} - f(d_n, k_{d_n}))^+, c_n \Big), ..., \\ & \Big((\rho_\Delta^1 - f(\Delta, 1))^+, ..., (\rho_\Delta^{k_\Delta} - f(\Delta, k_\Delta))^+ \Big), \\ & \Big((\rho_\Delta^1 - f(\Delta, 1))^+, ..., (\rho_\Delta^{k_\Delta} - f(\Delta, k_\Delta))^+ \Big) \Bigg] \end{aligned} \tag{4.28}$$

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 3 for (DP) now has the following form:

$$E_{t-1}^*(\overline{\boldsymbol{x}}) \leftarrow \min_{s \in \mathcal{A}(\overline{\boldsymbol{x}})} \left(P_{ower}\left(\frac{s}{1 - K_s \gamma} \right) + \sum_{\overline{\boldsymbol{x}}' \in \overline{\mathcal{X}}} P_t(\overline{\boldsymbol{x}}, s, \overline{\boldsymbol{x}}') E_t^*(\overline{\boldsymbol{x}}') \right) \tag{4.29}$$

Note that the speed $\frac{s}{1 - K_s \gamma}$ may not be directly available, but using the remark made in Section 4.5.1, 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:

$$K_s = \sum_{i=1}^{\alpha-1} k_i + \alpha \tag{4.30}$$

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

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 $\boldsymbol{x}_t$, instead of all their sizes. Therefore, the state becomes

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

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 $\boldsymbol{x}'_{t+1} = (w_{t+1}, k_{t+1}, \ell_{t+1})$ is such that:

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

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

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

In the processor sharing case, the additional time due to switching is γ per time unit. The Bellman equation is the same as Eq. (4.29) but replacing K_s by:

$$K'_s = \sum_i \frac{\mathbf{1}_{\{k_i>1\}}}{s} (s - w(i-1))^+ + \mathbf{1}_{\{k_1>1\}} \frac{w(1)}{s} \tag{4.31}$$

4.6 Conclusion

In this chapter, we showed how to select online 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 offline solutions on average, and outperforms classical online 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 chapter 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 offline 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. Most of the time this information is not available when jobs arrive: For example the exact job execution time (c) is only known after the job execution, and not at release time. That's why, in the next chapter, Chapter 5, we study a similar problem where the deadline of each job (d) is also revealed to the processor when it is released, however the *actual execution time* of each job is not known at release time but only when it finishes executing. Nevertheless we consider, in the next chapter, that we have some knowledge at release time on the execution time distribution and so on the worst case execution time (denoted W_{cet}). We take advantage of this information to improve the energy consumption in a more general case than in this chapter.

4.7 Appendix: Speed Decision and Changing Instants under the Optimal Policy

This appendix presents an analysis of the impact of the speed change discretization on the optimality of the solution. It is proved in Section 4.7.1 and 4.7.2 that without loss of optimality one can consider only integer speed decision and changing instants.

4.7.1 Speed Decision Instants

The energy consumed by the optimal policy when the speed decision instants can occur at each instant is:

$$\begin{aligned}
\mathbb{E}\int_0^T P_{ower}(s^*(t))dt &= \mathbb{E}\sum_{t=0}^{T-1}\int_t^{t+1} P_{ower}(s^*(t))dt \\
&\geq \mathbb{E}\sum_{t=0}^{T-1} P_{ower}\left(\int_t^{t+1} s^*(t)dt\right) &(4.32)\\
&= \mathbb{E}\sum_{t=0}^{T-1} P_{ower}\left(s^{mean}(t)\right) &(4.33)\\
&= \mathbb{E}\sum_{t=0}^{T-1} \beta P_{ower}(s_1^*(t)) + (1-\beta)P_{ower}(s_2^*(t)) &(4.34)
\end{aligned}$$

where s^{mean} is the average speed used between t and $t+1$. Equation (4.32) comes from Jensen inequality for the convex function P_{ower}. $s_1^*(t), s_2^*(t)$ in (4.34) are the neighboring speeds in $\mathcal{S}$ of the average speed $s^{mean}(t)$.

Ineq. (4.33) implies that the policy π^{mean} that chooses constant speed $s^{mean}(t)$ during each time interval t to $t+1$, for all $t \in \mathbb{N}$, is better or equal in energy consumption than this one that can change decision instants at any time $t \in \mathbb{R}$.

Furthermore, as deadlines and release times are integers, there is no random innovation between t and $t+1$, and so no new constraints on the system. $s^{mean}(t)$ is not necessarily available, that's why the power consumption of this speed is fixed to $P_{ower}(s_1^*(t)) + (1-\beta)P_{ower}(s_2^*(t))$. With these two remarks, the policy that uses speeds s_1^* and s_2^* is feasible, and by Ineq. 4.32 consumes less energy than the optimal policy s^*. As policy that uses s^* is optimal, then Ineq. (4.32) is in reality an equality.

Without loss of optimality, we can so consider that speed decision changes occur only at integer instants. Even if all parameters are integers (except the speed changing instants), this does not prevent the remaining work function of having some non integer values, and to have a bounded state space for the MDP. To satisfies that conditions, we have to prove that we can reduce to the case where speed changing instants are integers, and it is the subject of the next section.

4.7.2 Speed Changing Instants

After analysing the speed decision instants and showing that they can be consider as integer without loss of generality, now we will focus on the speed changing instants. It may seem natural to consider as speed decision instants, that speed changing instants are integer, because all job arrivals occur at integer times. However, this overlooks the fact that allowing the processor to change its speed at any time t in $\mathbb{R}$ gives to the processor a new degree of freedom that could be beneficial in terms of energy consumption.

Here is a simple example illustrating this fact: If the processor can use speed $s = 0$ over the sub-interval $[t, t+1/2]$ and speed $s' = 1$ over the sub-interval $[t+1/2, t+1]$, then, in total over the interval $[t, t+1]$, this can be seen as a processor using speed $s" = 1/2$, with an energy consumption $P_{ower}(s)/2 + P_{ower}(s')/2$. This combination is not possible if the processor can only change its speed at integer times and this new possibility may help to decrease the energy consumption. In this appendix, we will show that this is not the case.

To do so, let us consider a processor using the finite set of speeds $\mathcal{S}$ with respective powers $\{P_{ower}(s), s \in \mathcal{S}\}$ that may change its speed at any time in $\mathbb{R}$. Its minimal expected energy consumption when starting in state x is denoted $E^{*,\mathcal{S},\mathbb{R}}(x)$.

A priori such a processor is more capable than a processor using the same finite set of speeds $\mathcal{S}$ that may only change its speed at times in $\mathbb{N}$, whose minimal expected energy loss when starting in state $\boldsymbol{x}$ is now denoted $E^{*,\mathcal{S},\mathbb{N}}(\boldsymbol{x})$ (in the chapter, this was simply denoted $E^*(\boldsymbol{x})$). For all $\boldsymbol{x}$,

$$E^{*,\mathcal{S},\mathbb{R}}(\boldsymbol{x}) \leq E^{*,\mathcal{S},\mathbb{N}}(\boldsymbol{x}).$$

We will show in the following that equality always holds when the set of speeds is consecutive.

Theorem 4.1. *If the set $\mathcal{S}$ is made of consecutive speeds (*i.e.*, $\mathcal{S} = \{0, 1, 2, \ldots, s_{\max}\}$), then there is no energy gain for the processor to use non-integer speed changing instants: for all $\boldsymbol{x}$, $E^{*,\mathcal{S},\mathbb{R}}(\boldsymbol{x}) = E^{*,\mathcal{S},\mathbb{N}}(\boldsymbol{x})$.*

Proof. The proof is done with a simplified case where job arrivals have the same independent distribution at each time step, so that we can consider w as the state instead of the more general state $\boldsymbol{x} = (\ell, w)$ used in the chapter. The proof is essentially the same, as explained at the end of the proof. To keep notations simple, we also skip the indices $\mathcal{S}, \mathbb{N}$ in $E^{*,\mathcal{S},\mathbb{N}}$ in the proof up to the last part of the proof.

Let us consider that the set $\mathcal{S}$ of processor speeds is consecutive and that the processor can change its speed at times $t \in \mathbb{N}$ as well as at times $t + 1/2$. We will show that this does not bring any energy gain.

The time horizon is T and the minimal energy at time t under state w can be decomposed into two actions (taken at times t and $t + 1/2$):

$$E^*_{t+\frac{1}{2}}(w) = \min_{v \in \mathcal{A}_2(w)} \left(\frac{P_{ower}(v)}{2} + \sum_a P_t(a) E^*_{t+1} \left(\mathbb{T}_2 \left(w - \frac{v}{2} \right)^+ + a \right) \right) \tag{4.35}$$

$$E^*_t(w) = \min_{u \in \mathcal{A}_1(w)} \left(\frac{P_{ower}(u)}{2} + E^*_{t+\frac{1}{2}} \left(\mathbb{T}_2 \left(w - \frac{u}{2} \right)^+ \right) \right) \tag{4.36}$$

where $\mathcal{A}_2(w) = \{s \in \mathcal{S} : s \geq 2w(1)\}$, $\mathcal{A}_1(w) = \mathcal{S}$, $P_t(a) = \mathbb{P}(a_t(.) = a)$ and the operator $\mathbb{T}_2(f)$ only shifts the function f by $1/2$: $\mathbb{T}_2(f(x)) = f(x + 1/2)$.

This is a similar dynamic programming equation as used in Algorithm 3, where we take into account the fact that there are no arrival at time $t + 1/2$, and a modified admissibility condition on the speeds: to meet all deadlines, the speed at time $t + 1/2$ must execute all the work with deadline $t + 1$, hence the speed u must be larger than $2w(1)$ while the speed chosen at time t does not have any constraint: no job has a deadline at time $t + 1/2$. These two equations show that the new state space should also include the step functions with step sizes in $\mathbb{N}/2$.

By replacing the value of $E^*_{t+1/2}$ in the second equation, one gets E^*_t as a function of E^*_{t+1}:

$$\begin{aligned} E^*_t(w) &= \min_{u \in \mathcal{A}_1(w)} \left(\frac{P_{ower}(u)}{2} \right. \\ &\quad \left. + \min_{v \in \mathcal{A}_2(w)} \left(\frac{P_{ower}(v)}{2} \sum_a P_t(a) E^*_{t+1} \left(\mathbb{T}_2 \left(\mathbb{T}_2 \left(w - \frac{u}{2} \right)^+ - \frac{v}{2} \right)^+ + a \right) \right) \right) \\ &= \min_{u \in \mathcal{A}_1(w), v \in \mathcal{A}_2(w)} \left(\frac{P_{ower}(u)}{2} + \frac{P_{ower}(v)}{2} \right. \\ &\qquad \left. + \sum_a P_t(a) E^*_{t+1} \left(\mathbb{T} \left(w - \frac{u+v}{2} \right)^+ + a \right) \right) \end{aligned}$$

where we have used the distributivity of $+$ over $\max$ to get the second line.

This says precisely what was asserted without proof at the beginning: changing speed at half times is equivalent to choosing half speeds at integer times.

The first property that one can get from the last equation is the following: The speeds u, v achieving the $\min$ are such that $|u - v| \leq 1$. Indeed, if $|u - v| > 1$, then one can choose $u', v' \in \mathcal{A}_1(w), \mathcal{A}_2(w)$ such that $|u' - v'| \leq 1$ and $u + v = u' + v'$. By convexity of P_{ower}, $P_{ower}(v)/2 + P_{ower}(u)/2 \geq P_{ower}(v')/2 + P_{ower}(u')/2$ so that the choice u', v' is better than the choice u, v.

With no loss of generality, we will assume in the following that either $u = v$ (in which case we are back to an integer speed) or $v = u + 1$.

A second property is that both optimal speeds u and $u + 1$ are admissible in state w: If $u + 1 \in \mathcal{A}_2(T_2(w - u/2)^+)$, then $u + 1 + u \geq 2w(1)$. This implies $u \geq w(1) - 1/2$, so that $u \geq w(1)$ because both u and $w(1)$ are integers (and of course $u + 1 \geq w(1)$).

We are now ready for the proof, that holds by backward induction on t. Let us prove the two following properties simultaneously:

(P1) For all w with integer steps sizes, $E_t^*(w)$ is obtained by using integer speeds only.

(P2) For all w, a and all $u \in \mathcal{A}(w)$, then using $v = u + 1$,

$$E_t^*\left(\left(w - \frac{u+v}{2}\right)^+ + a\right) = \frac{1}{2}E_t^*\left((w-u)^+ + a\right) + \frac{1}{2}E_t^*\left((w-v)^+ + a\right).$$

Both properties are obviously true at time T where there is nothing to prove. Now, let us prove (P1) at time t:
Under state w with integer steps, let us consider the randomized policy that chooses at time t, speed $u \in \mathcal{A}(w)$ with probability $1/2$ and speed $u+1$ with probability $1/2$ and is optimal from time $t+1$ on.

The energy of this policy is

$$\begin{aligned} E_t^r(w) &= \frac{1}{2}\left(P_{ower}(u) + \sum_a P_t(a)E_{t+1}^*\left(\mathbb{T}(w-u)^+ + a\right)\right) \\ &\quad + \frac{1}{2}\left(P_{ower}(u+1) + \sum_a P_t(a)E_{t+1}^*\left(\mathbb{T}(w-u-1)^+ + a\right)\right) \\ &= \frac{P_{ower}(u)}{2} + \frac{P_{ower}(u+1)}{2} + \sum_a P_t(a)E_{t+1}^*\left(\mathbb{T}\left(w - \frac{u+u+1}{2}\right)^+ + a\right) \end{aligned}$$

where (P2) at time $t+1$ is used for states $(\mathbb{T}(w-u)^+ + a, \mathbb{T}(w-u-1)^+ + a, \mathbb{T}(w - \frac{u+u+1}{2})^+ + a)$ to get the second inequality.

This says that this randomized policy has the same energy cost as the policy that uses speed $(u+u+1)/2$ at time t. The theory of Markov decision processes says that there exists an optimal policy that does not randomize. Here, this implies that there exists an optimal policy at time t that uses an integer speed. This is exactly (P1).

As for (P2), we first notice that the arrival of jobs a can be included in the state w for simplicity. Therefore, let us consider two states with integer step sizes, $w_2 := (w-u+1)^+$ and $w_1 := (w-u)^+$ at time t. Using (P1), the optimal speed used in both states are integers. Let us denote by σ_1 the optimal speed used in state w_1.

Since $w_2 \geq w_1$ point-wise, then by monotony of the total energy with respect to the state, by using the same reasoning as in Proposition 4.5 and an induction on t, the optimal speed σ_2 in state w_2 is higher than σ_1: $\sigma_2 \geq \sigma_1$. We further claim that $\sigma_2 \leq \sigma_1 + 1$. We show this by contradiction: assume that $\sigma_2 = \sigma_1 + k$, with $k \geq 2$. Convexity of the power implies

$$P_{ower}(\sigma_1 + k) - P_{ower}(\sigma_1 + 1) \geq P_{ower}(\sigma_1 + k - 1) - P_{ower}(\sigma_1).$$

Since $\sigma_2 = \sigma_1 + k$ is optimal for w_2, we get

$$\begin{aligned} &P_{ower}(\sigma_1 + k) + \sum_a P_t(a)E_{t+1}^*\left(\mathbb{T}(w-u+1-\sigma_1-k)^+ + a\right) \\ &< P_{ower}(\sigma_1 + 1) + \sum_a P_t(a)E_{t+1}^*\left(\mathbb{T}(w-u+1-\sigma_1-1)^+ + a\right). \end{aligned}$$

Together with the previous inequality this implies

$$P_{ower}(\sigma_1 + k - 1) + \sum_a P_t(a) E^*_{t+1}\left(\mathbb{T}\left(w - u - \sigma_1 - k + 1\right)^+ + a\right) \tag{4.37}$$

$$< P_{ower}(\sigma_1) + \sum_a P_t(a) E^*_{t+1}\left(\mathbb{T}\left(w - u - \sigma_1\right)^+ + a\right). \tag{4.38}$$

The first term is the energy cost of using speed $\sigma_1 + k - 1$ in state w_1. The second term is the energy cost of using speed σ_1 in state w_1. This inequality contradicts the optimality of σ_1. Therefore, $\sigma_2 \leq \sigma_1 + 1$.

Now, let us compute the optimal speed σ_3 in the "middle" state $w_3 = (w - \frac{u+u-1}{2})^+$. By monotonicity, $\sigma_1 \leq \sigma_3 \leq \sigma_2$. Therefore, there only exists two possibilities:

- If $\sigma_1 = \sigma_2$ then $\sigma_3 = \sigma_1$ and

$$\begin{aligned}
E^*_t(w_3) &= P_{ower}(\sigma_1) + \sum_a P_t(a) E^*_{t+1}\left(\mathbb{T}\left(w - \frac{u+u-1}{2} - \sigma_1\right)^+ + a\right) \\
&= P_{ower}(\sigma_1) + \sum_a P_t(a) E^*_{t+1}\left(\mathbb{T}\left(w - \frac{(u+\sigma_1)+(u+\sigma_1)-1}{2}\right)^+ + a\right) \\
&= \frac{1}{2}\left(P_{ower}(\sigma_1) + \sum_a P_t(a) E^*_{t+1}\left(\mathbb{T}(w - u - \sigma_1)^+ + a\right)\right) \\
&\quad + \frac{1}{2}\left(P_{ower}(\sigma_1) + \sum_a P_t(a) E^*_{t+1}\left(\mathbb{T}(w + 1 - u - \sigma_1)^+ + a\right)\right) \\
&= \frac{1}{2} E^*_t(w_1) + \frac{1}{2} E^*_t(w_2)
\end{aligned}$$

  where the third equality comes from (P2) at time $t+1$.

- If $\sigma_2 = \sigma_1 + 1$ then by using the same argument to prove that $\sigma_1 \leq \sigma_2 \leq \sigma_1 + 1$, then the optimal speed in state w_3 is $\sigma_3 = (\sigma_1 + \sigma_2)/2$. In this case,

$$\begin{aligned}
E^*_t(w_3) &= \frac{1}{2} P_{ower}(\sigma_1) + \frac{1}{2} P_{ower}(\sigma_2) \\
&\quad + \sum_a P_t(a) E^*_{t+1}\left(\mathbb{T}\left(w - \frac{u+u-1}{2} - \frac{\sigma_1 + \sigma_1 + 1}{2}\right)^+ + a\right) \\
&= \frac{1}{2} P_{ower}(\sigma_1) + \frac{1}{2} P_{ower}(\sigma_2) + \sum_a P_t(a) E^*_{t+1}\left(\mathbb{T}(w - u - \sigma_1)^+ + a\right) \\
&= \frac{1}{2} E^*_t(w_1) + \frac{1}{2} E^*_t(w_2).
\end{aligned}$$

This shows that changing speeds at half times does not help. A straightforward generalization says that changing speeds at times $t \in \mathbb{N}/2^i$ will not help either for any i. By continuity of the total energy with respect to the speed function, this shows that changing speeds at times $t \in \mathbb{R}$ will not help either: $E^{*,\mathcal{S},\mathbb{R}}(w) = E^{*,\mathcal{S},\mathbb{N}}(w)$.

Finally, as announced at the beginning of the proof, one can take into account the more detailed state $\boldsymbol{x} = (\ell, w)$ instead of w. The proof is the same up to one harmless modification: Replace everywhere w by (ℓ, w) and

$$\sum_a P_t(a) E^*_{t+1} \left(\mathbb{T}(w-v)^+ + a\right)$$

by

$$P_t(0)E^*_{t+1}\left(\ell+1, \mathbb{T}(w-v)^+\right) + \sum_{a\neq 0} P_t(a)E^*_{t+1}\left(1, \mathbb{T}(w-v)^+ + a\right).$$

□

When the available speeds do not form a consecutive set, it is possible that all the optimal speed schedules change speed at non integer times. Here is a simple example. Consider the degenerated case where there is a single arrival at time $t = 0$ with probability one, of a job of size 4 with deadline 3: $w_0 = 0$, $P_0((0, 4, 3)) = 1$ and $P_t(a) = 0$ for all $t > 0$ and all $a \neq 0$.

If $\mathcal{S} = \{0, 1, 3\}$ (non-consecutive), then all optimal speed schedules must use speed $s = 3$ during $1/2$ a unit of time before time 3, speed 1 during $5/2$ units of time before time 3, and then speed 0 from time 3 on. So at least one speed change must occur at a non-integer time.

As a side note, if the set of speeds were consecutive: $\overline{\mathcal{S}} = \{0, 1, 2, 3\}$, then all optimal speed schedules would use speed 2 during one time unit, speed 1 during 2 time units and speed 0 from time 3 on. This is achievable with speed changes occurring at integer times.

In the following, we show that if $\mathcal{S}$ is not consecutive, it is always possible to go back to the consecutive case with integer speed changing instants by interpolating the power function.

Theorem 4.2. *If the set $\mathcal{S}$ is not consecutive, the optimal speed policy can be constructed using integer speed changing instants under an augmented consecutive set of speeds and then using V_{dd}-hopping (defined in the proof).*

Proof. Let $\overline{\mathcal{S}}$ be the extended set of speeds to all integer speeds below $s_{\max}$: $\overline{\mathcal{S}} = \{0, 1, 2, \ldots, s_{\max}\}$. To do this extension, we use the same strategy as in Section 3.4.3 in Chapter 3.

First, we assign to each non available integer speed a power consumption by using a linear interpolation. More precisely, for each $s < s_{\max}$ and $s \notin \mathcal{S}$, let $s_1, s_2 \in \mathcal{S}$ be the two neighboring available speeds such that $s_1 < s < s_2$. Therefore, s can be seen as a convex combination of s_1 and s_2:

$$s = \beta s_1 + (1-\beta)s_2, \text{ with } \beta = \frac{s_2 - s}{s_2 - s_1}. \tag{4.39}$$

We define the power consumption of s as:

$$P_{ower}(s) = \beta P_{ower}(s_1) + (1-\beta) P_{ower}(s_2). \tag{4.40}$$

Once this is done for each non available speed, we can solve the problem over $\overline{\mathcal{S}}$ with integer speed changing instants (the unavailable speeds being seen as available with the power cost defined in Eq. (4.40)). According to our notation, the optimal energy when starting in x is $E^{*,\overline{\mathcal{S}},\mathbb{N}}(x)$. The optimal speed policy with integer speed changing instants are denoted $\{s^*(t)\}_{t\in\mathbb{N}} \in \{0, 1, 2, \ldots, s_{\max}\}$.

The following transformation is done at each integer time step $t \in \mathbb{N}$ (this is called V_{dd}*-hopping* in the following). In the time interval $[t, t+1)$, if the optimal speed $s^*(t)$ was not originally available ($s^*(t) \notin \mathcal{S}$), then it is replaced by its two neighboring available speeds s_1 and s_2 over sub-intervals $[t, t+\beta)$ and $[t+\beta, t+1)$ respectively. Since the deadlines are integers, no job will miss its deadline during the interval $(t, t+1)$. Fig. 3.4 in Section 3.4.3 in Chapter 3 illustrates this decomposition.

This new policy only uses speeds in $\mathcal{S}$ but contains speed changes at non-integer times. We denote by $E^{\text{VDD},\mathcal{S},\mathbb{R}}(\boldsymbol{x})$ its energy consumption.

Since the power cost $P_{ower}(s^*(t))$ is a linear interpolation of the power cost of the neighboring available speeds s_1 and s_2, the energy consumption over the interval $[t, t+1)$ is the same using speed $s^*(t)$ on $[t, t+1]$ and using the neighboring speeds s_1 and s_2 over the two sub-intervals $[t, t+\beta]$ and $[t+\beta, t+1]$. This also means that the total energy consumption is the same before and after using V_{dd}*-hopping* :

$$E^{\text{VDD},\mathcal{S},\mathbb{R}}(\boldsymbol{x}) = E^{*,\overline{\mathcal{S}},\mathbb{N}}(\boldsymbol{x}).$$

On the one hand, Theorem 4.1 states that, with consecutive speeds, integer speed changing instants minimize the total energy consumption. In other words, this can be written

$$E^{*,\overline{\mathcal{S}},\mathbb{R}}(\boldsymbol{x}) = E^{*,\overline{\mathcal{S}},\mathbb{N}}(\boldsymbol{x}).$$

On the other hand, the optimal solution only using the subset composed by the available speeds must use at least as much energy as when all the intermediate speeds are available. This implies

$$E^{*,\overline{\mathcal{S}},\mathbb{R}}(\boldsymbol{x}) \leq E^{*,\mathcal{S},\mathbb{R}}(\boldsymbol{x}).$$

Putting everything together yields the following sequence of inequalities:

$$E^{\text{VDD},\mathcal{S},\mathbb{R}}(\boldsymbol{x}) \geq E^{*,\mathcal{S},\mathbb{R}}(\boldsymbol{x}) \geq E^{*,\overline{\mathcal{S}},\mathbb{R}}(\boldsymbol{x}) = E^{*,\overline{\mathcal{S}},\mathbb{N}}(\boldsymbol{x}) = E^{\text{VDD},\mathcal{S},\mathbb{R}}(\boldsymbol{x}).$$

This shows that $E^{\text{VDD},\mathcal{S},\mathbb{R}}(\boldsymbol{x}) = E^{*,\mathcal{S},\mathbb{R}}(\boldsymbol{x})$. This equality says that the V_{dd}*-hopping* policy is optimal. This optimal policy is an easy patch over the optimal policy with integer decision time, using the extended set of speed. $\square$

Corollary 3. *The optimal policy with integer speed changing instants and using speeds in the consecutive set* $\mathcal{S} = \{0, 1, \ldots, s_{\max}\}$ *is dominant over all policies with continuous decisions times, continuous speeds in the interval* $[0, s_{\max}]$ *and interpolated powers. Using our previous notation, this can be written: For all state* $\boldsymbol{x}$*,* $E^{*,\{0\ldots s_{\max}\},\mathbb{N}}(\boldsymbol{x}) = E^{*,[0,s_{\max}],\mathbb{R}}(\boldsymbol{x})$.

Proof. Recall that the interpolated power of any speed $s \in [0, s_{\max}]$ is $P_{ower}(s) = \beta P_{ower}(s_1) + (1-\beta)P_{ower}(s_2)$, where s_1 and s_2 are the two neighboring speeds of s in $\mathcal{S}$, as in Eq. (4.40). Under this power function, $E^{*,\{0\ldots s_{\max}\},\mathbb{N}}(\boldsymbol{x}) \leq E^{*,[0,s_{\max}],\mathbb{R}}(\boldsymbol{x})$ by definition.

Let $s^*(t)$ be the optimal policy in continuous time with continuous speeds in $[0, s_{\max}]$. Then, starting in x,

$$\begin{aligned}
E^{*,[0,s_{\max}],\mathbb{R}}(\boldsymbol{x}) &= \mathbb{E} \int_0^T P_{ower}(s^*(t))dt \\
&= \mathbb{E} \sum_{t=0}^{T-1} \int_t^{t+1} P_{ower}(s^*(t))dt \\
&\geq \mathbb{E} \sum_{t=0}^{T-1} P_{ower} \left(\int_t^{t+1} s^*(t)dt \right) && (4.41) \\
&= \mathbb{E} \sum_{t=0}^{T-1} \beta P_{ower}(s_1^*(t)) + (1-\beta) P_{ower}(s_2^*(t)) && (4.42) \\
&\geq E^{*,\{0\ldots s_{\max}\},\mathbb{R}}(\boldsymbol{x}) && (4.43) \\
&= E^{*,\{0\ldots s_{\max}\},\mathbb{N}}(\boldsymbol{x}), && (4.44)
\end{aligned}$$

where Eq. (4.41) comes from Jensen inequality for the convex function P_{ower} and the fact that there is no random innovation between times t and $t+1$; where $s_1^*(t), s_2^*(t)$ in Eq. (4.42) are the neighboring speeds in $\mathcal{S}$ of the average speed $\int_t^{t+1} s^*(t)dt$; and Theorem 4.1 is used to finish the proof in Eq. 4.44. □

Theorems 4.1 and 4.2 are valid whatever the horizon time T, as a consequence there are still satisfied in the infinite case problem.

5 Online Minimization: Statistical Knowledge with *Non-Clairvoyant* Jobs

After studying the *Clairvoyant* case situation in Chapter 4, in this chapter, we focus on the *Non-Clairvoyant* situation. The difference with the previous chapter is the information we have on the active jobs. In *Non-Clairvoyant* case, the active jobs have known deadlines at release time, but job *actual execution times* are only known at completion time. By considering that we know the distribution of the execution time at release time, we can deploy some algorithms that lead to energy minimization. Studying *Non-Clairvoyant* jobs has been already done in the literature, that's why in the first section, Section 5.1, we present the state of the art of the energy minimization on *non-clairvoyant* jobs.

This chapter is based on [GGP19c], currently submitted to an international conference.

5.1 State of the Art

As explained in the introduction to the chapter, *single processor hard real-time energy minimization problem* consists in choosing, for each job released in the system, a processor speed to execute this job, such that all jobs meet their deadline and such that the total energy consumed by the processor is minimized.

In the following we recall the difference between the *offline* and the *online* case, and then we explore in detail the existing work on the main point of this chapter: the *non-clairvoyant* case.

In the *online case*, only the jobs released before t or at t are known at t. We further distinguish the *clairvoyant online case*, where the characteristics of each job (deadline and execution time) are revealed at release time. 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 [YDS95]. These speed policies have been compared with the optimal offline solution by Bansal et al. in [BKP07], who also computed their competitive ratio. Further improvements have been proposed in [LY05] and [BKP07], and in Chapter 4.

In the *non-clairvoyant online case*, the deadline of each job is revealed to the processor when it is released, but the *actual execution time* of each job is only known when it finishes executing; only a worst case execution time (denoted W_{cet}) is known at release time. This case has been first investigated by Lorch and Smith in [LS01; LS04] 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. Lorch and Smith demonstrate that the optimal speed policy, resulting in the minimal expected energy consumption, is the one that *procrastinates*. This means that the processor speed increases gradually as the job progresses, until it terminates its execution. A continuous evolution of the speed being unrealistic, a heuristics is also provided, which approximates the optimal solution with a piece-wise linear function having a limited number of speed changes.

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 [Xu+04]. The authors have proposed Practical PACE (PPACE), a fully polynomial time approximation scheme ε-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 [BS09]. Third, Zhang et al. have proposed the Optimal Procrastinating Dynamic Voltage Scaling algorithm (OPDVS) under the form of a constrained optimization problem [Zha+05].

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 work have focused on the constrained framework of a *frame based multi-task model* [RMM02; GK03], 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) [Zha+05], while Xu et al. have proposed a Hybrid Dynamic Voltage Scaling algorithm (HDVS), hybrid in the sense that it addresses both intra-task DVS and inter-task DVS [XMM07]. The (HDVS) algorithm is a fully polynomial time approximation scheme ε-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.

In the online sporadic case, Gaujal et al. have tackled uniprocessor real-time systems where each job is modelled by three random variables, its release time, its exact execution time, and its deadline. Using the knowledge of the probability distribution of the job features (release times, execution times, and deadlines), an optimal online speed policy is computed [GGP17], based on a Markov Decision Process [Put05].

Because Gaujal et al. assume that the execution times are exact, (PACE) is more general. However, (PACE) is only valid for a single job, while Gaujal et al. consider a finite or an infinite set of jobs, where several jobs can be active at the same time. Actually, Lorch and Smith also proposed a multi-job extension of their speed policy [LS04], but by considering each job *independently* and simply adding the speeds obtained for each job in isolation, resulting in a sub-optimal speed selection.

The goal of this chapter is to extend the *non-clairvoyant online case* to a general set of jobs, possibly infinite. Each job is defined by its release time, execution time, and deadline. The job characteristics are only known as probability distributions, and several jobs can be active at the same time. In this respect we generalize both (PACE) and the work of Gaujal et al [GGP17].

We build a Markov Decision Process (MDP) that computes the *optimal* speed of the processor, 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 offline, we claim it does not hamper the online 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 proposed in [LS04].

The chapter is organized as follows. Related work having already been covered in the introduction, we formalize the problem in Section 5.2. Then we build our (MDP) solution in Section 5.3. We compare our solutions with previous work in Section 5.4. We perform numerical experimentation on synthetic and real-life benchmarks in Section 5.5. Finally we give concluding remarks in Section 5.6.

5.2 Formalization

5.2.1 System Model

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. The inter-arrival time is bounded by L. From the τ_i values, we can reconstruct the *release time* r_i of each job J_i. The two others parameters are the *execution time* c_i, bounded by W_{cet}, and the *relative deadline* d_i, bounded by Δ. 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 $\mathbb{N}$: $\mathcal{S} = \{s_1 = 0, s_2, \dots, s_k = s_{\max}\}$. For any job J_i, the execution time c_i is the execution time at the nominal speed 1 (the slowest possible speed). In our formalization, we therefore interpret this nominal execution time as the *size* of J_i, *i.e.*, the total work quantity that the processor must achieve to finish J_i: at speed 1, it will take c_i time units to complete the job. At any time t, the remaining work necessary to complete J_i is its size c_i minus the work quantity already spent by the processor on J_i. If at t this remaining work is c'_i and if the processor is executing J_i at speed $s(t)$, then at $t+1$ the new remaining work for J_i will be $c'_i - s(t)$ (or 0 if $s(t) \geq c'_i$). Processor speed changes may occur only at integer times. The cost of speed switching is assumed to be null. Our model can be generalized with a non-null speed switching thanks to the technique introduced in [GGP17].

The dynamic evolution of the system is as follows. It starts a time 0 with an empty state. The first job J_1 arrives at time $r_1 > 0$. 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 until reaching the time horizon T (or forever in the case of an infinite number of jobs). As EDF is optimal for *feasibility* (see Appendix A.1), this scheduling policy will be used as explained in Chapter 2.

In Section 5.4 we will compare our speed policy with (OA) and (PACE). (OA) also is a global policy, so we will use EDF too 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 for all the jobs active at time t), so in this case we will 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*). As defined in Chapter 2, the total energy consumption is:

$$E = \sum_{t=0}^{T} P_{ower}(s(t)) \tag{5.1}$$

while the long run energy consumption *average* is:

$$g = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} P_{ower}(s(t)) \tag{5.2}$$

The 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 online. 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 τ_i, c_i, d_i are viewed as *random variables*, for which we have *probability distributions* (because, for example, they have been estimated by numerous executions of the system).

5.2.2 State Space

In this section, the state $\boldsymbol{w}$ will be described, and as the information we have on jobs are different, we have to adapt the state definition. In this chapter the information we don't have, in comparison with Chapter 4, is the execution time of jobs, but we know at each instant the work quantity already executed of each specific jobs. The solution is so to replace in the state described in Chapter 4, d_i by the work quantity already executed. Let us described in detail the state and state space description.

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 sizes, 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. At any time $t \in \mathbb{N}$, the system state is made of:

- The set of active jobs (jobs released before t, whose execution is not yet finished at t).
- The relative deadline of each active job.
- The amount of work already done by the processor on each of the active jobs.

Formally, the state space of the system is defined as:

Definition 5.1 (System state)**.** *The system state at time t, denoted $\mathbf{w}(t) \in \mathcal{W}$, is composed of two elements:*

- *ℓ: the time elapsed since the latest job arrival.*
- *$\mathbf{x}$: the list of active jobs, sorted by their deadlines (ties are sorted by job release times). Each job J_i is characterized 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 of jobs $\mathbf{x}$. So $\mathcal{W} = \{1, \dots, L\} \times \mathcal{X}$, where L is the maximal inter-arrival time between any two jobs.

5.2.3 State Space Evolution

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

In the following we simply put into formula the two possible changes in the state space: Either some jobs will be completed during the current time interval $(t, t+1]$ by the processor, running at its current speed $s(t)$. This will remove the first jobs in the list $\mathbf{x}(t)$ since the processor is assumed to execute jobs in the EDF order (EDF being optimal for feasibility). Or some new jobs will arrive at time $t+1$, which must be inserted in the list $\mathbf{x}(t+1)$.

We introduce two operators that will be used to formalize the effect of jobs arrivals and completions.

Definition 5.2. *Let $\mathbf{x} = [(e_1^{\mathbf{x}}, d_1^{\mathbf{x}}), ..., (e_n^{\mathbf{x}}, d_n^{\mathbf{x}})]$ and $\mathbf{y} = [(e_1^{\mathbf{y}}, d_1^{\mathbf{y}}), ..., (e_n^{\mathbf{y}}, d_n^{\mathbf{y}})]$ be two job lists. We define two binary operators:*

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

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

$$\boldsymbol{x}_t = [(e_1^t, d_1^t), ..., (e_n^t, d_n^t)] \tag{5.3}$$

Furthermore, let us suppose 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:

$$[(e_{u+1}^{t+1}, d_{u+1}^{t+1}), ..., (e_n^{t+1}, d_n^{t+1})] \tag{5.4}$$

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 quantity executed over 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 *Shift* 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 is the work quantity executed on the not finished $(u+1)^{th}$ job:

$$\mathit{Shift}_u(\boldsymbol{x}_t, r) = (e_{u+1}^t + r, d_{u+1}^t - 1) \times (e_i^t, d_i^t - 1)_{\{i>u+1\}}$$

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 such as:

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

In summary, to compute the next job list $\boldsymbol{x}_{t+1}$ from the job list $\boldsymbol{x}_t$ and the processor speed s_t, we perform the following steps:

1. We compute the number of jobs executed u and the work executed on the unfinished job r, under the processor speed s_t on $\boldsymbol{x}_t$.

2. Then as the time goes on and the processor runs, the present jobs and their relative deadlines evolve, hence the *Shift* operator, which performs the following modifications:

    - Due to the processor execution, we remove from the ordered list $\boldsymbol{x}_t$ the u executed jobs, and we add the executed work quantity r to the $(u+1)^{th}$ job of $\boldsymbol{x}_t$.

    - Due to the time progress, we shift by one unit the relative deadline of the remaining jobs at $t+1$.

3. The last point is to merge the new list of jobs $a(t+1)$ with the remaining jobs at $t+1$, such as they are ordered by deadline (EDF policy) and then by jobs arrival date. We therefore obtain the next job list $\boldsymbol{x}_{t+1}$.

5.2.4 Construction of the Transition Probability Matrix

This is the most important part to construct a Markov Decision Process and build the optimal speed policy. In this part we construct the transition matrix $\mathbf{P}_t((\boldsymbol{x}, \ell_1), s, (\boldsymbol{y}, \ell_2))$ that gives the probability to go from state $(\boldsymbol{x}, \ell_1)$ to state $(\boldsymbol{y}, \ell_2)$ over time step $(t, t+1]$, when the processor uses speed s.

In the following, we give an explicit construction of $\mathbf{P}_t((\boldsymbol{x}, \ell_1), s, (\boldsymbol{y}, \ell_2))$ 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]

Before unveiling the probability formula, we introduce in the following some notations for the distributions of the job features:

- The i.i.d. inter-arrival times have a common distribution denoted θ:

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

- The i.i.d. sizes' distribution is denoted σ:

$$\forall 1 \le c \le W_{cet},\ \sigma(c) = \mathbb{P}(c_i = c) \text{ for any job } J_i$$

- The i.i.d. deadlines' distribution is denoted δ:

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

When all jobs are i.i.d., the transition probability $\mathbf{P}_t((\boldsymbol{x}, \ell_1), s, (\boldsymbol{y}, \ell_2))$ does not depend on t and can be decomposed in several parts, depending on the number of completed jobs. If we denote by $Q((\boldsymbol{x}, \ell_1), u, s, (\boldsymbol{y}, \ell_2))$ the probability to go from state $(\boldsymbol{x}, \ell_1)$ to $(\boldsymbol{y}, \ell_2)$ under speed s while u jobs are completed during a time step, then:

$$\mathbf{P}_t((\boldsymbol{x}, \ell_1), s, (\boldsymbol{y}, \ell_2)) = \sum_{u=\left\lfloor \frac{s}{W_{cet}} \right\rfloor}^{\min(s, \mathrm{nb}(\boldsymbol{x}))} Q((\boldsymbol{x}, \ell_1), u, s, (\boldsymbol{y}, \ell_2)). \tag{5.5}$$

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

The probability Q to go from state $(\boldsymbol{x}, \ell_1)$ to state $(\boldsymbol{y}, \ell_2)$ with u jobs completed during a time step can be further decomposed into the probability $P_{exec}(\mathbf{k}, \boldsymbol{x}, u, r)$ that u jobs are completed under speed s with a partial execution of r units of work on the next job, and the probability $P_{arrival}$ that a set $a(t)$ of jobs arrives in interval $(t, t+1]$ (independent of P_{exec}). Let us introduce the random variable k_i, the remaining size of job J_i and $\mathbf{k} = (k_1, \ldots, k_u)$.

- If $u = \mathrm{nb}(\boldsymbol{x})$ (all jobs are completed), then

$$Q((\boldsymbol{x}, \ell_1), u, s, (\boldsymbol{y}, \ell_2)) = \sum_{k_1 + \cdots + k_u \le s} P_{exec}(\mathbf{k}, \boldsymbol{x}, u, 0) \times P_{arrival}((\boldsymbol{x}, \ell_1), u, (\boldsymbol{y}, \ell_2)) \tag{5.6}$$

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

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

$$Q((\boldsymbol{x},\ell_1),u,s,(\boldsymbol{y},\ell_2)) = \sum_{k_1+\cdots+k_u=s-(e^{\boldsymbol{y}}_\alpha - e^{\boldsymbol{x}}_{u+1})} P_{exec}(\mathbf{k},\boldsymbol{x},u,e^{\boldsymbol{y}}_\alpha) \times P_{arrival}((\boldsymbol{x},\ell_1),u,(\boldsymbol{y},\ell_2)), \tag{5.7}$$

where α is the index of the first job in the list $\boldsymbol{y}$ with the same deadline as the first job in the list $Shift_u(\boldsymbol{x}, e^{\boldsymbol{y}}_\alpha - e^{\boldsymbol{x}}_{u+1})$. By definition, we have:

$$0 \leq e^{\boldsymbol{x}}_{u+1} \leq e^{\boldsymbol{y}}_\alpha \leq W_{cet} - 1$$

From now on, we compute each term P_{exec} and $P_{arrival}$:

▶ P_{exec} is the probability that the u first jobs of $\boldsymbol{x}$ are completed. It depends on (i) the probability of k_i, the remaining work executed during $(t, t+1]$, on job $i \in [1,u]$, and on (ii) 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 $u \neq 0$, with c_i the random variable that represents the size of job i:

$$\begin{aligned} P_{exec}(\mathbf{k},\boldsymbol{x},u,e^{\boldsymbol{y}}_\alpha) &= \left(\prod_{i=1}^{u} \mathbb{P}(c_i = k_i + e^{\boldsymbol{x}}_i \mid c_i > e^{\boldsymbol{x}}_i)\right) \mathbb{P}(c_{u+1} > e^{\boldsymbol{y}}_\alpha \mid c_{u+1} > e^{\boldsymbol{x}}_{u+1}) \\ &= \left(\prod_{i=1}^{u} \sigma(k_i + e^{\boldsymbol{x}}_i) / \sum_{k=e^{\boldsymbol{x}}_i}^{W_{cet}} \sigma(k)\right) \left(\sum_{k=e^{\boldsymbol{y}}_\alpha}^{W_{cet}} \sigma(k) / \sum_{k=e^{\boldsymbol{x}}_{u+1}}^{W_{cet}} \sigma(k)\right), \end{aligned} \tag{5.8}$$

And the second case is $u = 0$:

$$P_{exec}(\mathbf{k},\boldsymbol{x},0,e^{\boldsymbol{y}}_\alpha) = \left(\sum_{k=e^{\boldsymbol{y}}_\alpha}^{W_{cet}} \sigma(k) / \sum_{k=e^{\boldsymbol{x}}_{u+1}}^{W_{cet}} \sigma(k)\right). \tag{5.9}$$

▶ $P_{arrival}(\boldsymbol{x},u,\boldsymbol{y})$ 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:

$$a(t) = \boldsymbol{y} \ominus Shift_u(\boldsymbol{x}, e^{\boldsymbol{y}}_\alpha - e^{\boldsymbol{x}}_{u+1}) \tag{5.10}$$

Eq. (5.10) returns a list of new jobs present in the new list of jobs $\boldsymbol{y}$ and not present in the previous list $\boldsymbol{x}$, so they must be fresh arrivals. Under the form of a list, $a(t) = \{(0, d_i)\}_{i=1..n}$.

To compute the probability $P_{arrival}$, 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 jobs of deadline d_j. It satisfies $n = n_1 + \ldots + n_k$;

- $M(\boldsymbol{x})$ 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 $s_{\max}\Delta/W_{cet}$ to guarantee feasibility), the maximal number of job arrivals depends on the previous state $\boldsymbol{x}$. This is why $M(\boldsymbol{x}) = B - \mathit{nb\_job}(\mathit{Shift}_u(\boldsymbol{x}, e^{\boldsymbol{y}}_{\alpha} - e^{\boldsymbol{x}}_{u+1}))$ is the maximal number of jobs that can arrive at time $t+1$.

We note $\mathcal{S}_{\boldsymbol{x}}$, the set of possible successor of $\boldsymbol{x}$.

The function $P_{arrival}$ 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 $P_{arrival}$=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 $\ell_2 \neq \ell_1 + 1$, then $P_{arrival} = 0$.
- If $\ell_2 = \ell_1 + 1$, then $P_{arrival} = \mathbb{P}(a(t)) = 1 - \frac{\theta(\ell_1)}{\sum_{i=\ell_1}^{L}\theta(i)}$.
- If $\ell_1 = L$, then $P_{arrival} = 0$ (some work must arrive when the maximal inter-arrival time is reached)

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

- If $\ell_2 \neq 1$, then $P_{arrival} = 0$ (the time elapsed since the latest arrival must be reset to 1).
- If $\ell_2 = 1$, then using W_{cet} (the biggest job size), the general case for the probability $P_{arrival}$ is presented below $\forall n \in [1, M_n(x)]$.

Let us analyze the most general case: $a(t) \neq \emptyset$ and $\ell_2 = 1$. 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, as there is at least one arriving job ($a(t) \neq \emptyset$), it means that we have to consider the inter-arrival time ℓ_1, which is chosen among all the possible inter-arrival times, *i.e.* all values between ℓ_1 and L. This probability, which is the probability that the first job of $a(t)$ arrives, is $\frac{\theta(\ell_1)}{\sum_{i=\ell_1}^{L}\theta(i)}$. 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 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 we have:

$$\prod_{i=1}^{k} \delta(d_i)^{n_i} \tag{5.11}$$

But since jobs $(0, d_i)$ of same d_i are not ordered in $a(t)$, the product in Eq. (5.11) 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 take into account all possible combinations of jobs divided by all the combination for jobs of same deadline, because those are not ordered:

$$\frac{n!}{\prod_{i=1}^{k} n_i!} \tag{5.12}$$

These two analyses lead to the following job arrival probability for $a(t) \neq \emptyset$ and $\ell_2 = 1$, in the case where $n < M_n(\boldsymbol{x})$:

$$P_{arrival} = \frac{\theta(\ell_1)\,\theta(0)^{n-1}(1-\theta(0))}{\sum_{i=\ell_1}^{L}\theta(i)} \frac{n!}{\prod_{i=1}^{k} n_i!} \prod_{i=1}^{k} \delta(d_i)^{n_i} \tag{5.13}$$

In a second step, we consider that $n = M_n(\boldsymbol{x})$. So this situation simplifies Eq. (5.13) by removing the probability that there is not an extra $(|a(t)|+1)^{th}$ job:

$$P_{arrival} = \frac{\theta(\ell_1)}{\sum_{i=\ell_1}^{L}\theta(i)}\,\theta(0)^{n-1} \frac{n!}{\prod_{i=1}^{k} n_i!} \prod_{i=1}^{k} \delta(d_i)^{n_i} \tag{5.14}$$

Summary of the transition probability matrix construction: Putting together Eqs. (5.5), (5.6), (5.7), (5.8), (5.13) and (5.14) gives the formula to compute $\mathbf{P}_t()$ from the original distributions θ, σ, δ. This formula is combinatorial, but it is used only once and computed offline.

We present below the steps needed to compute one value of the probability matrix, *i.e.*, to go from state $\boldsymbol{x}$ to state $\boldsymbol{y}$ under speed s. As mentioned before, to compute it from the triplet $(\boldsymbol{x}, s, \boldsymbol{y})$, we have to study all the possible cases for the number of executed jobs of $\boldsymbol{x}$, denoted u. For all possible u, the steps of the algorithm for $(\boldsymbol{x}, u, \boldsymbol{y})$ are the following:

- Extract the first job of deadline d_{u+1} from $\boldsymbol{x}$.
- Compute the arrival jobs $a(t) = \boldsymbol{y} \ominus \mathit{Shift}_u(\boldsymbol{x}, r_\alpha)$.
- Compute the probability P_{exec}, which depends on u, the executed work for the first job of deadline d_{u+1} and the previous state $\boldsymbol{x}_t$.
- Compute the probability $P_{arrival}$, that depends on $a(t)$ value.

This construction of $\mathbf{P}_t()$ is also an *a posteriori* validation that $(\boldsymbol{x}(t), \ell_1), s(t)$ is a properly defined Markov Decision Process: $(\boldsymbol{x}, \ell_1)$ and s contain enough information to compute the probability of any next state $(\boldsymbol{y}, \ell_2)$.

5.3 Markov Decision Process

The energy consumption of the processor over one time interval $(t, t+1]$, when working at speed $s(t)$ is denoted $P_{ower}(s(t))$.

Computing at each time t the optimal speed s^* to minimize the expected energy consumption over T steps, namely $F_0(\boldsymbol{w}) = \mathbb{E}\sum_{u=0}^{T} P_{ower}(s(u))$ can be done by solving the following backward optimality Bellman equation [Put05].

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

$$F_t^*(\boldsymbol{w}) = \min_{s \in \mathcal{A}(\boldsymbol{w})} \left(P_{ower}(s) + \sum_{\boldsymbol{w}'} P(\boldsymbol{w}, s, \boldsymbol{w}') F_{t+1}^*(\boldsymbol{w}') \right) \tag{5.15}$$

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

$$\mathcal{A}(\boldsymbol{w}) = \left\{ s \in \mathcal{S}, s \geq \sum_{i:d_i=1} (W_{cet} - e_i) \right\} \tag{5.16}$$

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 [Put05].

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

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

$$g^* + h(\boldsymbol{w}) = \min_{s \in \mathcal{A}(\boldsymbol{w})} \left(P_{ower}(s) + \sum_{\boldsymbol{w}'} P(\boldsymbol{w}, s, \boldsymbol{w}') h(\boldsymbol{w}') \right). \tag{5.17}$$

Here again, the optimal speed $s^*(\boldsymbol{w})$ is any speed that achieves the $\min$ in the above equation.

The quantities g^* as well as the optimal speeds $s^*(\boldsymbol{w})$ can be computed offline using value iteration, whose complexity is quadratic 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 can be used to reduce the size of the state space.

Theorem 5.1. *Let us define B the maximal number of jobs that can be present in the buffer at a time instant, W_{cet} and Δ, the respective maximal job sizes and deadlines, then:*

1. *If $s_{max} \geq W_{cet}B$, 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 $s_{max} \geq W_{cet}$ then under the optimal speeds $s^*(\boldsymbol{w})$ all jobs will be completed before their deadlines.*

Proof. As B is the maximal number of jobs that can be present in the buffer at a time instant t and W_{cet} is the maximal job size, we can deduce that at each instant, the workload of the processor is at most $B * W_{cet}$. The worst case appears when all job deadlines are shortest, *i.e.* when job deadlines are 1. In this situation, a processor speed of $B * W_{cet}$ can execute all jobs before their deadlines, that's why if $s_{max} \geq W_{cet} * B$, all jobs can be executed before their deadline in all states that can be reached with positive probability.

In the particular case where jobs are released one at a time, the maximal work quantity that arrives at each time step is W_{cet}. If we use a maximal processor speed smaller than W_{cet}, we can be faced to a job of size W_{cet} and deadline 1, so deadline would be miss. However, if the maximal processor speed is faster than W_{cet}, it is possible to execute all jobs before their deadlines. Therefore if $s_{max} \geq W_{cet}$ then the optimal speed policy $s^*(\boldsymbol{w})$ will also execute all jobs before their deadlines.

$\square$

The algorithm of selecting speed value from (MDP) is as follows with σ^* the table of the optimal speed for each state:

Algorithm 6: Dynamic Programming processor speed choice algorithm

```
For Each  $t = 0 \ldots T-1$
    Update $\boldsymbol{x}_t$ according to the Section 5.2.3
    Set $s := \sigma^*[\boldsymbol{x}_t]$
    Execute the job(s) with earliest deadline
        at speed $s$ for one time unit
End
```

5.4 Comparative Analysis with Alternative Solution

We focus on two main alternatives to our solution ((OA) and (PACE), presented in full details in the section), proposed in the literature that 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. The work of Bini and Scordino [BS09] is only usable when the processor has two available speeds, and does not propose a clear generalization. The series of papers by Xu et al. [Xu+04; XMM05; XMM07] also provide efficient solutions to the energy minimization problem. They only deal with periodic tasks with a deadline equal to the period and identical periods (frame based model). Generalizing this to periodic tasks with arbitrary deadlines or to sporadic jobs does not seem realistic.

The first alternative (OA) is oblivious to size and deadline distributions. This online speed selection called Optimal Available (OA) was first presented in Yao & al [YDS95]. The second alternative uses a closed form solution for the optimal speed for each job independently. This is called (PACE) and was introduced in [LS01]. In the next two subsections, we will present these two algorithms and discuss their respective drawbacks in worst case scenarios. The comparison with our solution over a large set of job parameters will be the main focus of the following experimental section.

5.4.1 Optimal Available (OA) Speed Selection

The optimal available (OA) algorithm, introduced by Yao et al. [YDS95] is an online speed policy that chooses the speed $s^{(\text{OA})}(\boldsymbol{w}_t)$ at time t as follows. In any state $\boldsymbol{w}_t$, $s^{(\text{OA})}(\boldsymbol{w}_t)$ is the optimal speed in order to execute the current remaining work at time t, should all job sizes be equal to their W_{cet} 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 ${W_{cet}}^i - e_i$ (with our notation $x_t^{(\mathrm{OA})} = (e_i, d_i)_{i=1\ldots n}$), then:

$$s^{(\mathrm{OA})} = \max_{i=1\ldots n} \frac{{W_{cet}}^i - e_i}{d_i}. \tag{5.18}$$

Optimal Available (OA) Speed Selection Algorithm

We denote $\boldsymbol{x}_t^{(\mathrm{OA})}$ the state of the system under (OA) speed policy. The algorithm selecting the speed selected at time t by (OA) is as follows:

Algorithm 7: (OA) processor speed choice algorithm

```
For Each  $t = 0 \ldots T - 1$
    Update $\boldsymbol{x}_t^{(\mathrm{OA})}$ according to the Section 5.2.3.
    Compute $s^{(\mathrm{OA})}$ using Eq. (5.18).
    Execute the job(s) with earliest deadline at speed $s^{(\mathrm{OA})}$
      for one time unit
End
```

Example showing (OA) can be sub-optimal

(OA) has many good properties: It is obviously robust to changing job features and has a constant competitive ratio with an optimal offline solution [BKP07]. However, it is certainly sub-optimal, especially because it does not take advantage of jobs with small sizes.

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, then it will use according Algorithm 7, at each time $t = 0, 1, 2, 3$, the same speed: $s_t^{(\mathrm{OA})} = 1$.

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

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

On the other hand, in such an 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, using the same approach as in [XMM05]. Computing these optimal speeds boils down to a constrained convex minimization problem: Minimize $C(s(0)^2 + \frac{3}{4}s(1)^2 + \frac{2}{4}s(2)^2 + \frac{1}{4}s(3)^2)$ under the constraints $s(i) \geq 0, 0 \leq i \leq 3$ and $s(0) + s(1) + s(2) + s(3) = 4$. Using a Lagrange multiplier λ, the Karush-Kuhn-Tucker conditions are: $s(0) = \lambda$, $\frac{3}{4}s(1) = \lambda$, $\frac{2}{4}s(2) = \lambda$ and $\frac{1}{4}s(3) = \lambda$, under the constraint $s(0) + s(1) + s(2) + s(3) = 4$. This implies $s(0) = \frac{12}{25}, s(1) = \frac{16}{25}, s(2) = \frac{24}{25}, s(4) = \frac{48}{25}$ with the total expected energy:

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

The relative over-consumption of (OA) versus the optimal policy for this simple job, $(\mathbb{E}(E^{(\mathrm{OA})}) - \mathbb{E}(E^*))/\mathbb{E}(E^*)$, is just above 30%. The following section 5.4.2 introduces (PACE), a speed policy that is optimal as long as the state $\boldsymbol{x}$ contains a single job. In this example (PACE) will be optimal and will reduce the energy consumption by 30% over (OA).

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 Section 5.5.

5.4.2 (PACE) Speed Selection

Single Job Speed Selection

In [LS04], 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 one job $J = (c, d)$ is as follows, with F the cumulative distributed function of the size of J $(F(.) = \mathbb{P}(w \leq .))$.

$$s^{(\mathrm{PACE})}(e) = K(1 - F(e))^{-1/3} \tag{5.19}$$

The normalizing constant K is obtained by solving the following equation that makes sure that a job of maximal size W_{cet} is completed before d:

$$\int_0^{W_{cet}} \frac{1}{s^{(\mathrm{PACE})}(w)} \mathrm{d}w = d \tag{5.20}$$

(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. $\forall i \leq c \leq i+1$,

$$F(w) = (F_\sigma(i+1) - F_\sigma(i))F_\sigma(i)(w - i) + F_\sigma(i), \tag{5.21}$$

written under the form $F(w) = a_i w + b_i$ 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 K for a job with deadline d is:

$$K = \frac{3}{4d} \sum_{[i,i+1]\in[0,W_{cet}]} \frac{1}{a_i} \left[(1 - ia_i - b_i)^{4/3} - (1 - (i+1)a_{i+1} - b_{i+1})^{4/3}\right] \tag{5.22}$$

And so the speed for a job J with deadline d when e work has already been executed on J, is

$$s^{(\text{PACE})}(J(e,d)) = K\left[1-(a_i e + b_i)\right]^{-1/3} \text{ with } i \leq e < i+1 \tag{5.23}$$

The speed computed by Eq. (5.23) is not, in general, an integer. We therefore round it to the closest available speed from the set $\mathcal{S}$:

$$round(s,\mathcal{S}) = \underset{s' \in \mathcal{S}}{\operatorname{argmin}}\{|s-s'|\} \tag{5.24}$$

$$\tilde{s}^{(\text{PACE})}(J) = round(s^{(\text{PACE})}(J),\mathcal{S}) \tag{5.25}$$

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 active 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:

$$\min\left\{ s \in \mathcal{S} | s \geq \sum_{J_i \in \boldsymbol{x}_t} \tilde{s}^{(\text{PACE})}(J_i) \right\} \tag{5.26}$$

This set could be empty, meaning that the speed $s_{\max}$ is not high enough to execute all the jobs present in the system before their deadlines. To prevent this from occurring, we will add in the next section a feasibility condition.

(PACE) **Algorithm**

Since the state of our system $\boldsymbol{x}_t$ depends on the choice of the speed policy, we will denote by $\boldsymbol{x}_t^{(\text{PACE})}$ the state of the system under (PACE) speed policy. The algorithm of selecting the current speed under (PACE) speed policy is as follows:

Algorithm 8: (PACE) processor speed choice algorithm

```
for each $t = 0$ to $T-1$ do
    Update $\boldsymbol{x}_t^{(\text{PACE})}$ according to the Section 5.2.3;
    for each job $J_i \in \boldsymbol{x}_t^{(\text{PACE})}$ do
        Compute $s^{(\text{PACE})}(J_i(e_i,d_i))$ using Eq. (5.19);
    end for each
    if $d_i = 1$ then    // for schedulability reason
        $\tilde{s}^{(\text{PACE})}(J_i) = W_{cet} - e_i$;
    end if
    Execute each job of state $\boldsymbol{x}_t^{(\text{PACE})}$ at speed:
        $\tilde{s}^{(\text{PACE})}(J_i)$ for one time unit;
end for each
```

5.4.3 Suboptimality 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 k jobs $J_1, \dots J_k$ are released simultaneously, all with the same size ($c = 1$), with respective deadlines $1, 2, ..., k$. 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 $s^{(\text{PACE})}(J_i) = 1/i$, so the cumulative speed at time 0 is

$$\begin{aligned} s^{(\text{PACE})}(J_1) + ... + s^{(\text{PACE})}(J_k) &= 1 + 1/2 + ... + 1/k \\ &= H_k \approx \log k. \end{aligned}$$

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

$$\begin{aligned} \mathbb{E}(E^{(\text{PACE})}) &= C\left(H_k^2 + (H_k - H_1)^2 + ... + (H_k - H_{k-1})^2\right) \\ &= C\left(kH_k^2 + \sum_{i=1}^{k-1} H_i^2 - 2H_k\left(\sum_{i=1}^{k-1} H_i\right)\right) \\ &= C\left(kH_k\left(H_k - \frac{2(k-1)}{k}\right) + \sum_{i=1}^{k-1} H_i^2\right) \\ &= Ck(\log k)^2 + O(Ck\log k). \end{aligned}$$

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

$$\mathbb{E}(E^{(\text{OA})}) = Ck.$$

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

These two examples (subsections 5.4.1 and 5.4.3) show respectively that in some cases (PACE) behaves much better than (OA), and in some other cases, (OA) is better. The following numerical experiments show that under certain sets of job features, (OA) (resp. (PACE)) can be very close to the optimal speed as computed by our (MDP) algorithm while in other cases it can be very far from our optimal policy. While this large range of relative loss can sometimes be explained, in some other cases it is rather hard to understand the true cause of inefficiency of (OA) or (PACE) with respect to (MDP).

5.5 Numerical Experiments

5.5.1 Experimental Set-up

The experiments are done in two phases. The first part is offline and consists in computing the optimal speed policy with the (MDP) solution. We use the (MDP) algorithm described in this chapter to determine, for each state $\boldsymbol{x} \in \mathcal{X}$, the optimal speed to apply to the processor. These speeds are stored in a table.

In the second part, we simulate a sequence of jobs, with the probability distributions used to construct the (MDP). At each time instant, we use the previous table to determine the speed of the processor in the next time slot. For (OA) and (PACE), we compute the processor speed with Algorithm 7 and 8 described in Section 5.4 respectively.

Finally, we compare the mean energy consumption of each policy over several sequences of jobs with the same time horizon.

In a first set of experiments, we use synthetic benchmark by using all possible discrete size distributions over a fixed range, but we also analyze the effect of inter-arrival time and the deadline distribution over the performance of the three policies.

In a second set of experiments, we use our solution in a practical case: each job is a run of an edge detection algorithm. The runtime of these jobs are measured and the empirical distribution of the execution times of these jobs is used to assess the performance of the 3 speed policies.

Here are the results of all these tests in a nutshell: (MDP) outperforms (OA), when job are highly irregular, and it outperforms (PACE) when the number of pending jobs is often high. For sequences of jobs where a single job is pending at any time (for example for periodic tasks), the performance of (MDP) and (PACE) are close, both being optimal on average in that case. Differences are due to the time and speed discretization of (PACE) making it sub-optimal when the discretized solution is far from the ideal one.

In the next sub-section, the algorithm of online simulation is presented.

5.5.2 Online Simulation Algorithm

During the online simulation part, we run several sequences of jobs over a large time horizon T. During one simulation, a given sequence of jobs is generated using the distributions of the features (θ, σ, δ).

The processor speed policies (MDP), (PACE), and (OA) presented in Section 5.3 and 5.4 are used over the sequence of jobs, and the total energy consumption for each of these policies is computed. After running several simulations (usually 1000, to get a good confidence on the result), we compare the empirical mean energy consumption of the three policies.

At the beginning of each online simulation, for a given speed policy, we generate an inter-arrival time that follows the inter-arrival time distribution $\theta(t)$, conditioned to be in the interval $[1, L]$

so that a job can arrive at the earliest at the next time step (here $t = 1$). Then we decrease the current inter-arrival time of one unit of time when we move to the next time step. When the next release time is reached (unless the buffer is full) then a new job is generated: we generate its size according to the size distribution with support $[1, W_{cet}]$ and its deadline according to the deadline distribution, with support $[1, \Delta]$. As soon as the job is created then we generate another inter-arrival time that follows the inter-arrival time distribution over the interval $[0, L]$, this time. If this inter-arrival time is 0 and the buffer is not full, then we generate another job immediately, otherwise we go to the next time step. If the buffer is full, no other job can arrive at the current time-step, and we have to choose an inter-arrival time on the normalized inter-arrival time distribution conditioned on the interval $[1, L]$.

After this simulation step, the set $a(t)$ of job arriving at time $(t+1)^-$ have been generated. The processor speed is computed under state x_t and used to executed jobs present in the system state at time t. The next state x_{t+1} is computed as explained in Sec 5.2.3.

As job generations follow the same law for (MDP), (OA), and (PACE), this algorithm is applied for each policy, and then we can compare each mean energy value computed over these simulations.

5.5.3 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 Section 5.4.2.
- 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 Section 5.2.4). 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.

5.5.4 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) described in Algorithm 6, and compare it with the energetic performance of the two other policies (PACE) and (OA), described in Algorithm 8 and in Algorithm 7 respectively. In all these experiments, we analyze the relative over-consumption. The over-consumption of the policy (PACE) in comparison of the policy (MDP) is defined as follows:

$$\text{Over-consumption} = \frac{\text{Energy}_{(\text{PACE})} - \text{Energy}_{(\text{MDP})}}{\text{Energy}_{(\text{MDP})}} \tag{5.27}$$

The same formula is used for (OA).

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 $W_{cet} = 4$.
- Buffer size, as defined in Section 5.2.4, is set to $M_n = 3$.

In the two following tables, Table (5.1) and (5.2), we summarize the over-consumption obtained by the two policies (OA) and (PACE) for different inter-arrival time distributions, in the cases where the maximal inter-arrival time L is bounded by 1 (see Tab. (5.1)) and when L bounded by 4 with no multiple job arrivals (see Tab (5.2)).

Table 5.1.: Inter-arrival time distribution influence on the over-consumption of (OA) versus (MDP), and of (PACE) versus (MDP) on simulations with uniform deadline and Δ=3, and uniform size and W_{cet} =4, with $L = 1$

Inter-arrival time distribution		Over-consumption	
$\tau = 0$	$\tau = 1$	(PACE) versus (MDP)	(OA) versus (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 5.2.: Inter-arrival time distribution influence on the over-consumption of (OA) versus (MDP), and of (PACE) versus (MDP) on simulations with uniform deadline, with Δ=3, and uniform size, with W_{cet} =4, and $1 < L < 4$.

Inter-arrival time distribution				Over-consumption	
$\tau = 1$	$\tau = 2$	$\tau = 3$	$\tau = 4$	(PACE) versus (MDP)	(OA) versus (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%

From Table (5.1) and (5.2), one can notice that the better the knowledge of the inter-arrival time arrival, the worse the over-consumption of (OA) versus (MDP). The trend is the same for (PACE) policy.

The other aspect is the number of jobs present in the system at a time t. One can also notice that when there are several job in the buffer, (PACE) speed policy consumes a lot of energy in comparison to (MDP). This is due to the fact that (PACE) is only optimal for one job is been executed at each time. When no job arrives until the active job is completed, (PACE) consumes the same energy as (MDP).

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 (5.3) and 3 in Table (5.4). The size is uniform between 1 and $W_{cet} = 4$ and the maximal deadline $\Delta = 3$. We test our algorithm for different deadline distributions.

Table 5.3.: Deadline distribution influence on the over-consumption of (OA) versus (MDP), and of (PACE) versus (MDP) on simulations with uniform size, with W_{cet} =4, and fixed inter-arrival time $L = 1$.

Deadline distribution			Over-consumption	
$d = 1$	$d = 2$	$d = 3$	(PACE) versus (MDP)	(OA) versus (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 5.4.: Deadline distribution influence on the over-consumption of (OA) versus (MDP), and of (PACE) versus (MDP) on simulations with uniform size, with W_{cet} =4, and fixed inter-arrival time $L = 3$.

Deadline distribution			Over-consumption	
$d = 1$	$d = 2$	$d = 3$	(PACE) versus (MDP)	(OA) versus (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 (5.3), for the situation where there is one job that arrives at each time step, that if there are more short deadlines, then the over-consumption of (PACE) is less important. This is expected, because when there are more short deadlines, it means that there are potentially less jobs present in the buffer. We are getting closer to 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 (5.4) 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 probabilities distribution, whereas (OA), which is oblivious of the probabilities distribution, suffers from a significant energy over-consumption.

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 $W_{cet} = 4$.

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:

$$\begin{gathered}\mathbb{P}(C=\{1,2,3,4\})=\left\{\frac{i_1}{10},\frac{i_2}{10},\frac{i_3}{10},\frac{i_4}{10}\right\}\\ \text{with } i_1+i_2+i_3+i_4=10 \text{ and } i_4>0\end{gathered} \tag{5.28}$$

We compute the over-consumption of each policies (OA) and (PACE) in comparison with (MDP) for all possible distributions satisfying Eq. (5.28).

Impact of the size distribution on the over-consumption of (OA) **against** (MDP) Fig. 5.1 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 the W_{cet}. This is because (OA) is build as if each job had an size equal to its W_{cet}.

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 the over-consumption of (PACE) **against** (MDP) Fig. (5.2) 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 (5.2) 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 (5.2). 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

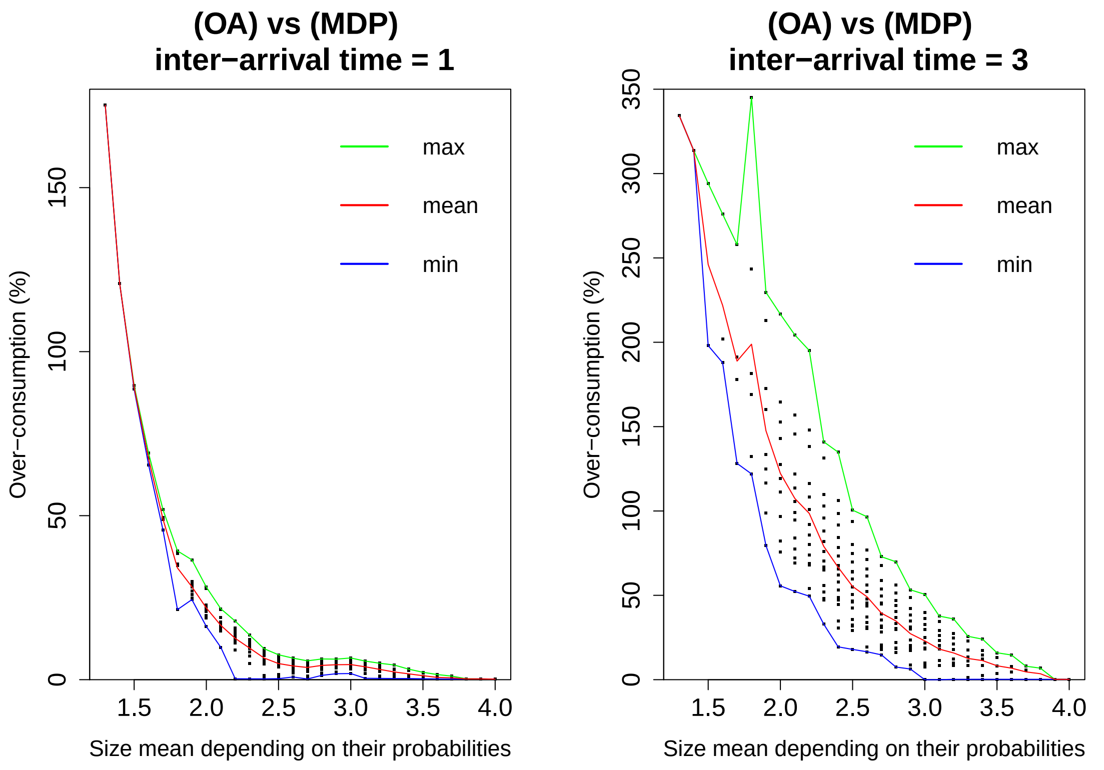


Figure 5.1.: Influence of the size distribution on the over-consumption of (OA) versus (MDP), with fixed jobs deadline $d = 3$, fixed inter-arrival time $L = 1$ (left) or $L = 3$ (right), and a fixed buffer size $B = 3$.

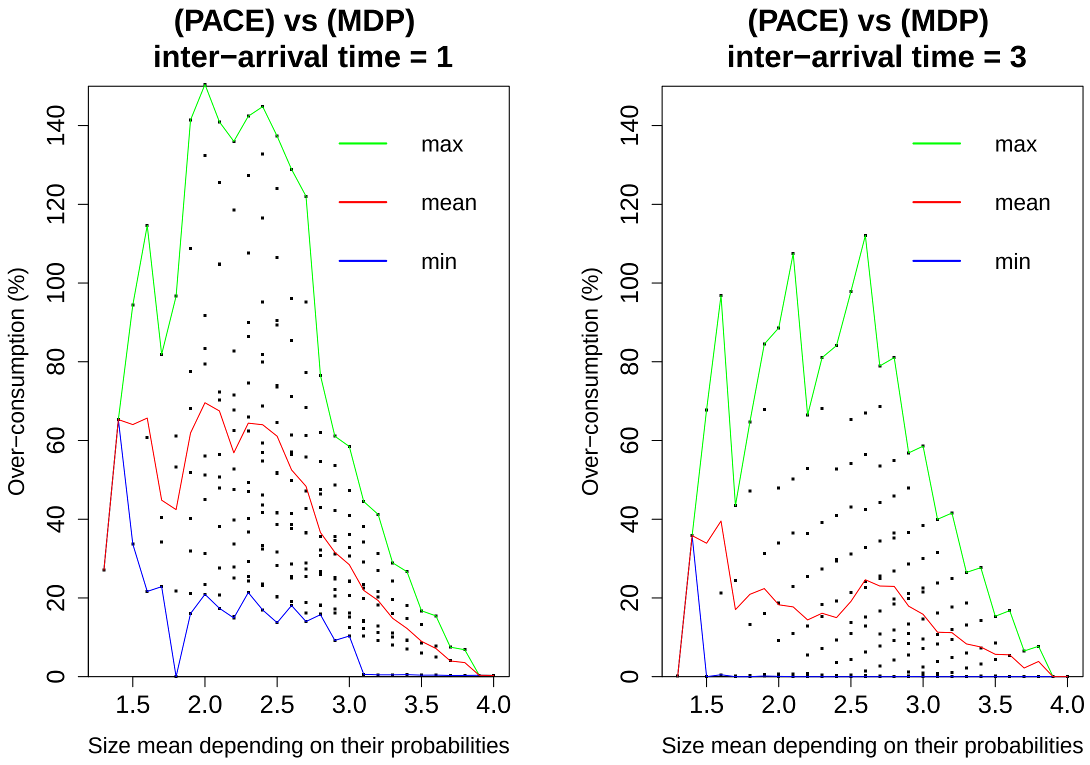


Figure 5.2.: Influence of the size distribution on the over-consumption of (PACE) versus (MDP), with fixed jobs deadline $d = 3$, fixed inter-arrival time $L = 1$ (left) or $L = 3$ (right), and a fixed buffer size $B = 3$.

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 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.

Test with an Edge Detection Algorithm

We tested our online 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 (see Figure 5.3).

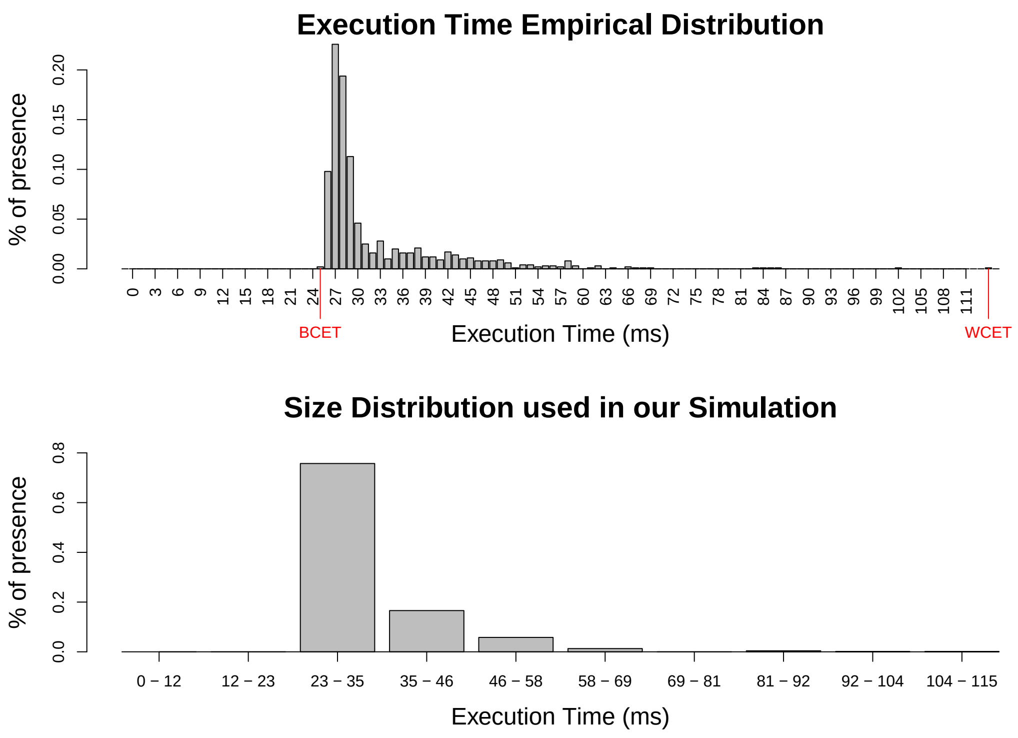


Figure 5.3.: Distribution of execution time for the edge detection algorithm over $1,000$ executions on a $1\,s$ video. The first barplot depicts the distribution of the execution time, and the second is the corresponding discretized distribution used to test (MDP), (OA) and (PACE).

Figure 5.3 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 Figure 5.3), we reduce it to 10 groups only by aggregating the values into groups as represented in the bottom part of Figure 5.3.

The overhead of (PACE) versus (MDP) on this example is 186%, while it is 106% for (OA) versus (MDP). Concerning (PACE), this is due to the discretization that is intrinsic to this algorithm. Concerning (OA), this is expected because the mean of the size distribution $(32.28\,ms)$ is significantly lower than the W_{cet} $(114\,ms)$.

5.6 Conclusion

We have proposed a Markov Decision Process (MDP) solution to compute the optimal online speed policy for the *single processor hard real-time energy minimization problem*. 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 online solution on average, and can be very attractive when the mean value of the execution time distribution is far from the W_{cet}, and/or when the statistical knowledge on the jobs' features is accurate.

As in Chapter 4, since the time and space complexity of our algorithm is exponential in the job deadlines, it will be interesting to find some methods to reduce the state space size, because it limits for now the applicability of our solution. A potential solution would be to reduce the state space by merging some "close" states of the (MDP).

Up to know, in Chapter 4 and 5, we do the assumption that for future jobs, we have a statistical information. Practically, this information is most often not available. Without these data, the only solution is to learn all the characteristics of job during the HRTS execution. In the following we present two chapters based on learning techniques, that determine online the optimal policy in the case of clairvoyant assumption for active jobs and no information for future jobs. The next chapter, Chapter 6 focus on specific learning: learning the distributions of the job features to learn the (MDP) parameters.

6 Online Minimization: Learning Transition Probability Matrix with Unknown Statistics

After considering the case where we have statistical information on active and/or future jobs, we will focus in this chapter on the case where we have no information on these jobs. Therefore, we have to use *learning techniques*. This chapter analyses one of the learning techniques we will see in this thesis. The other one will be present in Chapter 7. Let us begin with the state of the art and the problem statement.

6.1 State of the Art

Determining *on-line* the speed policy to apply to the processor can be done by learning the parameters of the Markov Decision Process (MDP) that guides our problem.

In this section, nothing is known on the arrival model of the jobs, nor on their characteristics. However, we know that an MDP is a model that can represent the system evolution. Therefore, our goal is to learn all the parameters of this MDP. This MDP will be described in Section 6.2. We are thus in the field of *model-based learning*.

During all these experimentations, we want to improve, as in the previous chapters (Chapter 3 to 5), the speed choices in order to tend to the average gain (here the minimal average energy cost). In this chapter, we focus on the *undiscounted case*, which is why we compute the *minimal average energy cost*. Nonetheless, some of our results are also valid in the discounted case (*i.e.*, with a discount $\gamma < 1$), and each time it is the case, it will be clearly stated.

To determine the MDP parameters, we first implement a *training part*, during which we learn the state space and the transition probability matrix. When the system evolves, we discover states, and we register all the possible jobs that arrive in the system. At some point in time, we stop this learning part. We then determine the probability of jobs arrival by averaging the number of job visits per learning step. We thus obtain all the parameters to solve our MDP. Finally, thanks to a Value Iteration algorithm (as in Chapter 4), we determine the optimal speed policy, which we apply to the processor starting from this time. Note that other algorithms can also be used to solve the MDP, such as the Policy Iteration algorithm. These classical algorithms, which compute the optimal average energy consumption, are provided for example in [Put05].

We only consider a discrete state space and discrete action space. The considered MDP is described in the next section.

6.2 Markov Decision Process

Let us start by providing an informal description of the model description, *i.e.*, the MDP. Then we observe the behavior of the system throughout its evolution. At each sample, we observe the job arrivals and their characteristics. A *sample* corresponds to the arrival of a set of jobs, possibly empty.

Jobs arrive during the execution of the system, and to decrease the present work quantity, the processor runs at different speeds during the time. More specifically, at each sample n, the processor has to use an available speed s_n to execute partially or totally jobs that are present in the system. The time evolution and the state description and shift are displayed in Fig. 6.1.

As the state space is independent of time (indeed at a certain time we have visited all the possible states), in the following we will only consider w, w', two states and analyze a transition from state w to state w' using speed s. Let us also define $\mathcal{S}_w$, the set of speeds that are both available in the processor specification and also that satisfy the feasibility of the process. In other words it means that speeds in $\mathcal{S}_w$ are all large enough to finish jobs that release at the next time unit.

Fig. 6.1 illustrates the evolution of our state from sample n to $n+1$.

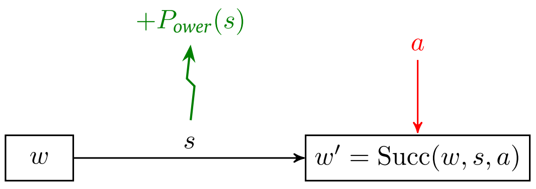


Figure 6.1.: Markov decision process evolution with job arrivals between sample n and $n+1$. At sample n, we are in state w and by using processor speed s, with job arrivals a, we go to state w'.

Formally, we consider the following MDP, denoted $(P_{ower}, \mathcal{W}, P, \gamma)$ in the following:

- The *state* of the system at sample n is the remaining work function w. The finite set of states is denoted $\mathcal{W}$.

- The only possible action that can be taken by the decision-maker (or learner, in the following) is the processor speed s_n selected at sample n among a finite set $\mathcal{W}$ (jobs are executed according to EDF scheduling).

- The transition from one state to the next depends on the speed s_n (that will reduce the remaining work of the jobs) and the next job arrivals. In the following, we denote by Succ the successor function: $w' = \mathrm{Succ}(w, s, a)$ returns the state w' succeeding to state w when the speed s is used over the time interval $[t, t+1]$ and the job arrivals at $t+1$ is a.

  When job arrivals follow a probability distribution ($P(a)$ denotes the probability that the next job arrival is a), this induces a *transition probability*, $P(w, s, w')$ from state w to state w' under speed s:
  $$P(w, s, w') = P(a) \quad \text{if} \quad w' = \mathrm{Succ}(w, s, a).$$

- The immediate cost after n samples is the power consumed by the processor at n, and is denoted $P_{ower}(s_n)$.

The average cost is the *undiscounted energy spent per time unit* and is defined by:

$$v = \mathbb{E}\left[\lim_{N\to\infty}\left(\frac{1}{N}\sum_{n=0}^{N} P_{ower}(s_n)\right)\right] \tag{6.1}$$

with N the total number of samples. This limit exists as soon as the MDP is weakly communicating (obviously true here because the state where no jobs are present can be reached with positive probability from any other state).

Alternatively, the average cost can be computed as the *limit of the total discounted cost*:

$$v = \lim_{\gamma\to 1}\mathbb{E}\left((1-\gamma)\sum_{n=0}^{\infty}\gamma^n P_{ower}(s_n)\right) \tag{6.2}$$

where $\gamma < 1$ is the discount. It follows that the discount cost for any initial state w is:

$$V^{\gamma}(w) = \mathbb{E}\left((1-\gamma)\sum_{n=0}^{\infty}\gamma^n P_{ower}(s_n)\right). \tag{6.3}$$

In the following we will focus on the *undiscounted* case. The discounted case will studied in details in Chapter 7. Some properties or theorems will be valid for the two situations, in which case it will be notified at the beginning of the section. Since the *discounted case* will be analyzed in details in Chapter 7, simulations in this chapter will be done for the *discounted case* to be able to compare the result of this chapter with the results of the next chapter.

6.3 Problem Statement

The goal of this chapter is to learn the speed policy to apply to the processor, that minimizes the average undiscounted energy, denoted v in Section 6.2, by considering, in contrary to the previous chapter, that job arrival distribution is unknown. In this case, we want to “build” these data by learning the job arrival distribution. The unknown information we seek to learn are the characteristics of the incoming jobs (sizes and deadlines) and also their arrival probabilities. We will discover these information while the jobs are arriving.

Therefore the transition probability matrix P of the MDP described in Section 6.2 is our unknown. To determine this matrix, we have to determine the probability $P(w, s, w')$ to go from state w to state w' under speed s. This value matches with the job arrival probability, because the transition probability matrix is a deterministic function in job distribution. As said before, this probability depends only on the workload arrivals at each sample n, denoted A_n, because of our job hypothesis, *i.e.*, the fact that there are i.i.d. for all job parameters. Furthermore, we assume a stationary assumption on the probability laws of the job release time. We define the workload arrival function A_n as follows:

$$A_n(.) = \sum_{i \,|\, r_i = n} c_i H_{r_i + d_i}(.) \tag{6.4}$$

A_n is given at each instant and with this new incoming information, we enhance the knowledge of the system. We call $\{A_n\}_{0,.,t}$ a *learning trajectory*, which is built progressively. These job arrivals are distributed by following the law of their features. After observing this function A_n, we can improve the value of the transition probability matrix P.

Practically, determining the speed schedule by learning the transition probability matrix proceeds in two phases:

1. *The learning phase*: In the first phase, we learn the transition probability matrix. In the sequel, we denote N the length of the learning phase. From 0 to N, a group of jobs arrives with a certain probability distribution at each time step, and we improve step after step the $\hat{P}_N$ matrix value. At time N, we therefore obtain $\hat{P}_N$.

2. *The application phase*: In the second phase, that is, at time N, we compute the optimal speed schedule of the MDP with transition probability matrix $\hat{P}_N$, and we apply this speed policy to the processor for any sample $n > N$.

To assess the resulting speed policy, we will compare it in terms of energy consumption, during the application phase, against the best speed policy that does not anticipate jobs arrival, that is, against the Optimal Available policy (OA) [YDS95].

6.4 Probability Transition Matrix Learning

We learn the transition probability matrix $P(w, s, w')$ and the state space $\mathcal{W}$ of the MDP. After this learning phase, we can determine the optimal speed policy by solving the MDP with a dynamic programming algorithm, such as the Value Iteration algorithm.

We study two different versions:

1. The *Asynchronous* version: At each iteration, the transition probability matrix is updated only for each triplet (w, s, w') that is visited. This version is called Asynchronous Probability Transition Matrix Learning.

2. The *Synchronous* version: At each iteration, the transition probability matrix is updated for all the possible triplets (w, s, w'). This version is called the Synchronous Probability Transition Matrix Learning (PL).

The synchronous version implies that, for each triplets (w, s, w'), we compute the following equations that depend of the characteristics of jobs arrival:

$$\forall w, w' \in \mathcal{W}, \forall s \in \mathcal{S}_{admissible}(w) \quad ,$$

$$\hat{P}_{n_i+1}(w, s, w') = \frac{n_i}{n_i+1}\hat{P}_{n_i}(w, s, w') + \frac{1}{n_i+1}, \text{ if } w' = \mathrm{Succ}(w, s, A_{n_i}) \tag{6.5}$$

$$\hat{P}_{n_i+1}(w, s, w') = \frac{n_i}{n_i+1}\hat{P}_{n_i}(w, s, w'), \text{ otherwise} \tag{6.6}$$

As explained before, A_n is the set of arrival jobs after n samples. The formula $w' = \mathrm{Succ}(w, s, A_n)$ corresponds to the next state function w' at the next sample and, according to Lemma 4.1 in Chapter 4, we have:

$$w' = \mathrm{Succ}(w, s, A_n) = \mathbb{T}\left[(w - s)^+\right] + A_n \tag{6.7}$$

Furthermore, we define the function $\mathrm{FirstPassing}(.)$, which returns the number of the first iteration where the state w is visited. As a result, n_i in Eq. (6.6) is defined as follows: $n_i = n - \mathrm{FirstPassing}(\boldsymbol{x})$, where n is the number of samples.

Once the probability transition matrix is learned, we use the Value Iteration algorithm to compute the optimal speed policy.

The asynchronous case obeys also to Eq. (6.6), however it modifies the state space and the probability matrix only for the current state w_n and the arbitrary (not necessarily optimal) speed s_n. The convergence time is longer, because we only update one triplet by timing iteration. In this chapter, we only present the Synchronous version (PL), because there exists other methods in the literature to solve the asynchronous part, which are not based on the same idea (more precisely, they are based on upper-confidence bound methods [AO06] and on Thomson sampling [ORR13]).

6.5 Learning Probability Matrix Algorithm (PL)

In this section, we describe the Synchronous Learning Probability Matrix Algorithm (PL). The line that depends on the system behavior is typeset in blue italic font. Here we gather the characteristics of the jobs that arrive at the time step $n + 1$.

The Asynchronous Learning Probability Matrix Algorithm is based on the same idea, except that we update only one triplet (w, s, w') at each time step. The loop in line 4 is suppressed and the $\hat{P}$ value update is done only for the current state w with the value corresponding to the previous speed policy, *i.e.*, only for one speed choice s. As a consequence, there is only one successor state w', and line 9 is executed only once for each iteration of the while loop. The speed choice at each time step is made such that the energy consumption is minimized. Then, by relying on a softmax computation[1], we choose a non-optimal speed to visit all the state space.

[1]`https://en.wikipedia.org/wiki/Softmax_function`

Algorithm 9: Synchronous Learning Probability Matrix Algorithm (PL)

```
 1: $\hat{P}_0(w,s,w')$ is set to an arbitrary value (say 0) for all $w$, $s$ and $w'$
 2: $Nvisit(w,s,w') \leftarrow 0$ for all $w$, $s$ and $w'$        % Visit counter for triplet $(w,s,w')$
 3: while $n < N$ do
 4:   for all state $w$ and all admissible speed $s$ do
 5:     Get the new incoming jobs $A_{n+1}$
 6:     $W \leftarrow \mathrm{Succ}(w,s,A_{n+1})$
 7:     for all state $w'$ do
 8:       if $W = w'$ then
 9:         $\hat{P}_{n+1}(w,s,w') \leftarrow \frac{Nvisit(w,s,w')}{Nvisit(w,s,w')+1}\hat{P}_n(w,s,w') + \frac{1}{Nvisit(w,s,w')+1}$
10:       else
11:         $\hat{P}_{n+1}(w,s,w') \leftarrow \frac{Nvisit(w,s,w')}{Nvisit(w,s,w')+1}\hat{P}_n(w,s,w')$
12:       end if
13:     end for
14:   end for
15: end while
```

Once the matrix $\hat{P}_n$ has been learned and constructed with Algorithm 9, we use the Value Iteration algorithm to compute the optimal speed schedule of the (MDP) that uses the transition probability matrix $\hat{P}_n$.

This raises the following question: Is the $\hat{P}_n$ matrix returned by Algorithm 9 close enough to the actual transition probability matrix P, so as to be able to compute the optimal speed policy? The next section will answer partially this question by unveiling a convergence criterion.

6.6 Convergence Criterion

In this section, we establish a convergence criterion for the learning phase, based on our estimate on the probability transition matrix. The convergence criterion presented in Prop. 6.1 will be valid for the two cases, undiscounted and discounted. Let v^* be the average cost of the optimal policy using the true distribution of jobs with transition matrices P, and let $\hat{v}_n$ be the average cost of the optimal policy using the estimate distribution of jobs after n samples, with transition matrices $\hat{P}_n$. Then a bound on $\hat{v}_n - v^*$ can be obtained by using perturbation analysis of Markov Chains.

The optimal cost vector is denoted by $V^{*,\gamma}$ and the estimated cost vector by $\hat{V}_n^{\gamma}$.

Proposition 6.1. *The error $\hat{v}_n - v^*$ (or $\|\hat{V}_n^{\gamma} - V^{*,\gamma}\|$ in the discounted case) is smaller than ε with high probability if n, the duration of the learning period, satisfies:*

$$n \geq \frac{1.66\, R_{\max}\, K}{\varepsilon^2}, \tag{6.8}$$

where $R_{\max}$ is the energy cost of the maximal speed, $K = \frac{1}{(1-\gamma)}$ in the discounted case and $K = \frac{1}{p_0^{\Delta}}$ in the undiscounted case (with p_0 the probability of no job arrival and Δ the maximal deadline).

Proof. Let us first consider the discounted case. By definition, for any state w,

$$\begin{aligned}\hat{V}_n^\gamma(w) &= \min_{s\in\mathcal{S}_{possible}(\boldsymbol{x})}\left\{(1-\gamma)P_{ower}(s)+\gamma\sum_{w'\in\mathcal{W}}\hat{P}_n(w,s,w')\hat{V}_n^\gamma(w')\right\}\\ &\leq \left\{(1-\gamma)P_{ower}(s^*)+\gamma\sum_{w'\in\mathcal{W}}\hat{P}_n(w,s^*,w')\hat{V}_n^\gamma(w')\right\}\end{aligned}$$

where s^* is the optimal speed under the exact distribution. Therefore, by denoting $P(s)$ the matrix $P(.,s,.)$, one gets

$$\begin{aligned}\|\hat{V}_n^\gamma - V^{*,\gamma}\| &\leq \gamma\|\hat{P}_n(s^*)\hat{V}_n^\gamma - P(s^*)V^{*,\gamma}\| && (6.9)\\ &\leq \gamma\|\hat{P}_n(s^*)\hat{V}_n^\gamma - \hat{P}_n(s^*)V^{*,\gamma}+\hat{P}_n(s^*)V^{*,\gamma}-P(s^*)V^{*,\gamma}\| && (6.10)\\ &\leq \gamma\left(\|\hat{V}_n^\gamma - V^{*,\gamma}\| + \|V^{*,\gamma}\|\|\hat{P}_n(s^*)-P(s^*)\|\right) && (6.11)\end{aligned}$$

Eq. (6.11) yields:

$$\begin{aligned}\|\hat{V}_n^\gamma - V^{*,\gamma}\| &\leq \frac{1}{1-\gamma}\|\hat{P}_n(s^*)-P(s^*)\|.\|V^{*,\gamma}\|\\ &\leq \frac{R_{\max}}{1-\gamma}\|\hat{P}_n(s^*)-P(s^*)\| && (6.12)\end{aligned}$$

where $R_{\max}$ is the energy cost of the maximal speed.

The undiscounted case is more complicated. By definition, $v^* = \mu^* P_{ower}$ with μ^* the stationary probabilities under the optimal policies for the exact distribution. Using the speed policy of the exact distribution under the estimated transition matrix gives a larger average cost: $\hat{v}_n \leq \mu_n P_{ower}$, where μ_n is the stationary probabilities of matrix $\hat{P}_n(s^*)$.

Let $p_0 > 0$ be the exact probability that no job arrives at the current sample. Then, from each state w, the state $(0,0,\dots,0)$ is reached after Δ steps with probability larger than p_0^Δ. This implies that $\mu^*(0,0,\dots,0) > p_0^\Delta$. Using the results and terminology presented in [IM95], the *absolute stability* of P can be written as $\|\mu^* - \mu_n\| \leq K\|\hat{P}_n(s^*)-P(s^*)\|$ with $K \leq \|A^{-1}\|$. Here A is the non-singular matrix $I-(P_{(0\dots0)})^\Delta$, where $P_{(0\dots0)}$ is the sub-matrix of P without its first row and first column. Since for any state $i \neq (0\dots0)$, we have $\sum_{j\neq(0\dots0)} P_{ij}^\Delta < 1-p_0^\Delta$, as explained above, then the matrix Q defined by $Q = \frac{1}{1-p_0^\Delta}(P^*_{(0\dots0)})^\Delta$ is a sub-stochastic matrix. The inverse of A is

$$A^{-1} = \sum_{n=0}^{\infty}(P_{(0\dots0)})^{\Delta n} = \sum_{n=0}^{\infty}(1-p_0^\Delta)^n Q^n$$

Since Q is sub-stochastic, we have $\|A^{-1}\| \leq 1/p_0^\Delta$. Finally,

$$\begin{aligned}\hat{v}_n - v^* &\leq \mu_n P_{ower} - \mu^* P_{ower}\\ &\leq \|\mu^* - \mu_n\|R_{\max}\\ &\leq \frac{R_{\max}}{p_0^\Delta}\|\hat{P}_n(s^*)-P(s^*)\|.\end{aligned}$$

In both cases, the error depends on the norm $\|\hat{P}_n(s^*) - P(s^*)\|$. Using a 95% confidence interval, we can determine a general bound on this norm, which depends only on the iteration number n.

By using the Central-Limit theorem and by defining σ the standard deviation of $\hat{P}_n$, and n the training duration, then $\hat{P}_n(.)$ satisfies the following equation:

$\forall w, w' \in \mathcal{W}^2, \forall s \in \mathcal{S},$

$$\mathbb{P}\left(P(w,s,w') - \frac{\phi^{-1}_{\mathcal{N}(0,1)}(0.95)\sigma}{\sqrt{n}} \le \hat{P}_n(w,s,w') \le P(w,s,w') + \frac{\phi^{-1}_{\mathcal{N}(0,1)}(0.95)\sigma}{\sqrt{n}}\right) \approx 0.95 \tag{6.13}$$

where $\phi_{\mathcal{N}(0,1)}(.)$ is the cumulative distribution function of the standard normal distribution.

Eq. (6.13) is valid for any triplet (w,s,w'), but some triplets can be visited more often during the learning phase (depending on the jobs arrival probability), so this equation can be satisfied very quickly in some cases. In other terms, it means that for some cell of the probability transition matrix $\hat{P}_n$, the convergence is reached quickly. We could have introduced a number n_i corresponding to the convergence bound of the n_i^{th} triplet. Instead, we prefer to keep the same bound n for all the triplets. Here n corresponds to $n = \max_{\forall w,s,w'} n_i$. It is the number of steps of the (PL) algorithm that we need to learn all the cells of the $\hat{P}_n$ matrix.

Therefore,

$$\mathbb{P}(\|\hat{P}_n(w,s,w') - P(w,s,w')\| \le \varepsilon) = 0.95 \Rightarrow \frac{\left(\phi^{-1}_{\mathcal{N}(0,1)}(0.95)\right)^2 \sigma^2}{\varepsilon^2} \le n \tag{6.14}$$

By numerical application, the convergence of P at $\varepsilon = 0.01$ at 95% is satisfied when:

$$2.576^2\,\sigma^2\,\varepsilon^{-2} \le n \tag{6.15}$$

Since each transition follows a binomial distribution, we have:

$$\sigma(w,s,w') = \sqrt{P(w,s,w')(1 - P(w,s,w'))} \le \frac{1}{2},$$

and therefore:

$$n \ge \frac{1.66}{\varepsilon^2} \tag{6.16}$$

This concludes the proof. □

It follows that, to satisfy $v_n - v^* \le 0.01$ in 95% of the simulations, the training period must use at least $n \ge 1.66\,K.10^8$ job arrivals. This imposes a very long training period to guarantee a small error in the speed policy. The numerical experiments reported in Section 6.9 show that, in most cases, a short training period is enough to obtain very good performances. This may come from two reasons: first, the theoretical bounds are not tight, and second, two quite different transition matrices may have similar optimal costs: in all cases, the lower the speed, the better the energy cost.

6.7 Comparison with Optimal Available policy (OA)

In this section, we show a comparison between the optimal policy σ^*, which gives the speed at a time instant if we have the exact value of the transition probability matrix P, and the Optimal Available (OA) speed policy in the undiscounted case [YDS95]. (OA) is interesting because it is an online speed policy that does not take into consideration statistical assumptions. As a consequence, our learning policy that learns the transition probability matrix has to beat (OA) in term of energy consumption. The solutions presented in Chapters 4 and 5 do consider some statistical knowledge, and the comparisons show that both outperform (OA) in terms of energy consumption.

This part focuses purely on the undiscounted case. The discounted case will be analysed in detail in Chapter 7.

Recall that (OA) is an online speed policy that chooses the speed $s^{(\mathrm{OA})}(w)$ in state w to be the minimal speed in order to execute the current remaining work w at sample n, should no further jobs arrive in the system. More precisely, in state w, (OA) uses the speed

$$s^{(\mathrm{OA})}(w) = \max_u \frac{w(u)}{u} \tag{6.17}$$

where $w(.)$ is the remaining work function.

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

Proposition 6.2. *Both in the finite or infinite case, the optimal speed policy σ^* satisfies*

$$\sigma^*(w) \geq s^{(\mathrm{OA})}(w) \tag{6.18}$$

for any state $w \in \mathcal{W}$, if the power consumption P_{ower} 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 $s^{(\mathrm{OA})}(w) = \max_u \frac{w(u)}{u}$ is reached for $u = 1$ (*i.e.*, $s^{(\mathrm{OA})}(w) = w(1)$), then $\sigma^*(w) \geq s^{(\mathrm{OA})}(w)$ by definition, because the set of admissible speeds $\mathcal{A}(w)$ only contains speeds larger than $w(1)$ (see Eq. (6.17)).

If the maximum is reached for $u > 1$, then $\mathcal{A}(w)$ may enable the use of speeds below $w(1)$.

Between the current time instant and u, some new job may arrive and therefore, the optimal policy should satisfy $\sum_{i=0}^{u-1} \sigma^*(wi) \geq w(u)$.

The convexity of the power function P_{ower} implies[2] that the terms of the optimal sequence $\sigma^*(w), \dots, \sigma^*(w_{u-1})$ must all be above the average value (which is larger than $w(u)/u = s^{(\mathrm{OA})}(w)$). In particular, for the first term, $\sigma^*(w) \geq s^{(\mathrm{OA})}(w)$.

Now, if the set of speeds is finite, then the optimal value of $\sigma^*(w)$ must be one of the two available speeds in $\mathcal{S}$ surrounding $\sigma^{(\mathrm{OA})}(w)$. Let s_1 and s_2 in $\mathcal{S}$ be these two speeds, *i.e.*, $s_1 < \sigma^{(\mathrm{OA})}(w) \leq s_2$, and assume again that the max in Eq. (6.17) is not reached for the first time step (*i.e.*, $u = 1$). If the smallest speed s_1 is chosen as the optimal speed, this implies that further choices for $\sigma^*(w_i)$ will have to be larger or equal to s_2, to compensate for the work surplus resulting from choosing a speed below $\sigma^*(w)$. This implies that it is never sub-optimal to choose s_2 in the first place (by convexity of the P_{ower} function).

This trajectorial argument is true almost surely, so that the inequality $\sigma^*(w) \geq s^{(\mathrm{OA})}(w)$ will also hold for the *expected* energy over both a finite or infinite time horizon. □

Prop. 6.2 allows us to decrease the state space study, because we can suppress all triplets (w, s, w') that use speed $s(w)$ below $s^{(\mathrm{OA})}(w)$.

6.8 Feasibility Condition

Recall Definition 2.2: The feasibility is the fact that the online speed policy misses no deadline by using only available speeds. In our case, the online speed policy is the one learned during the learning phase. The however issue is that, in the learning case, the job knowledge is incomplete, which raises the question of the feasibility condition. In fact, since the feasibility condition is independent of the job distribution, even if the system knowledge is incomplete, we can still keep the same feasibility condition. This is stated Theorem 6.1, which gives the value of $s_{\max}$ that ensures feasibility of the (PL) algorithm only in the undiscounted case.

This part analyses only the undiscounted case. The discounted case will be analysed in detail in Chapter 7

Theorem 6.1. *The feasibility of* (PL) *is ensured if and only if* $s_{\max} \geq C$.

Practically, before job executions we don't know if the policy on the system is feasible, but while the maximal job size satisfied $s_{\max} \geq C$, we are certain that the policy decided for the jobs execution is feasible. We therefore have a *dynamic* feasibility condition: As soon as a job size is above $s_{\max}$, we have a potential feasibility problem, otherwise the feasibility is guaranteed. Let us now know prove Theorem 6.1.

Proof. By definition, (PL) completes all the jobs before their deadline by construction: $\pi^{(\mathrm{PL})}(n) \geq w_n^{(\mathrm{PL})}(n+1)$. Therefore, (PL) is feasible if $\pi^{(\mathrm{PL})}(n) \leq s_{\max}$.

1. Case $s_{\max} < S$**:** In that case, no speed policy can guarantee the feasibility, since there always can exist a sequence of jobs, each with S work quantity, that arrive at each instant. In that situation,

[2]Actually, we use the fact that the sum $\sum_{i=0}^{u-1} P_{ower}(s)$ is Schur-convex when P_{ower} is convex (see [MO79]).

the mean work quantity that arrive is S, and since the maximal processor speed is strictly below S, we can not execute all the work quantity.

2. Case $s_{\max} \geq S$: To prove the result, we first modify the P_{ower} function as follows: For all speeds $s > s_{\max}$, we set $P_{ower}(s) = \infty$. For $s \leq s_{\max}$, the P_{ower} function remains unchanged. This modification is valid because the processor cannot use speeds larger than $s_{\max}$ anyway. Therefore, the energy consumption for such unattainable speeds can be arbitrarily set to any value. The benefit of using this modification is the following. Instead of forcing the speed to remain smaller than $s_{\max}$, we let the scheduler use unbounded speeds, but this incurs an infinite energy consumption. It follows that a test to check if a policy uses speeds larger than $s_{\max}$ is that its average energy consumption will be infinite.

Starting from an empty system with no pending job, *i.e.*, $w_0 = (0, 0, \dots, 0)$, we define the following naive policy $\tilde{\pi}$:

$$\forall t \in \mathbb{N},\ \tilde{\pi}(n) := c_n \quad \text{where } c_n = \sum_{J_i=(r_i,c_i,d_i)} \{c_i | r_i = n\}. \tag{6.19}$$

In other words, c_n is the amount of work that arrived at sample n, which is by definition less than S. The policy $\tilde{\pi}$ is feasible because it never uses a speed larger than $S \leq s_{\max}$ and all work is executed as fast as possible (within one time slot after its arrival). Furthermore, since for any n, $\tilde{\pi}(n) \leq S$, its long run expected energy consumption per time unit satisfies $Q_{\tilde{\pi}}(w_0) \leq P_{ower}(S)$.

The optimal policy, being optimal in energy, satisfies $Q_{\pi^{(\mathrm{PL})}}(w_0) \leq Q_{\tilde{\pi}}(w_0)$, hence $Q_{\pi^{(\mathrm{PL})}}(w_0) \leq P_{ower}(S)$. Therefore, (PL) is feasible by construction and never uses a speed larger than $s_{\max}$. □

6.9 Numerical Experiments

As described in previous sections (in particular Sections 6.3 and 6.4), we consider a training period over which the learning phase of (PL) is used with one or several typical job sequences to learn the optimal speed policy. Once the training period is over, after N samples, the learned policy is used *in production* to save energy in the deployed application.

During the training case, the performance metric is the length of the training period and the quality of the learned policy.

A key point is that the simulations are performed with a *reduced* set of processor speeds. More precisely, in state w we use the set $\mathcal{S}_{possible}(w) = \{s^{(\mathrm{OA})}(w), s^{(\mathrm{OA})}(w) + 1, s^{(\mathrm{OA})}(w) + 2\}$. The reason for this choice is that we expect (OA) to be "not too far" from the optimal speed policy. This is because (OA) is optimal when no further jobs arrive. Therefore, we expect the optimal speeds to be close to the speed chosen by (OA). This choice of reduced processor speeds, along with Prop 6.2, significantly decreases the duration of our experiments.

6.9.1 State Space Construction

Since we have no knowledge on the maximal job deadline Δ and of the maximal job size S beforehand, the state space $\mathcal{W}$ is not known in advance, so we cannot use directly the (PL) algorithm directly as displayed in Section 6.5. To solve this issue, we build the state space *progressively* and add all the possible states reached $w \in \mathcal{S}$ by using all the possible speed belonging to $\mathcal{S}_{possible}(.)$. When a group of jobs A_n arrives at time n, we update the current state w space by computing the successors of w for *all* states and *all* possible speeds with job arrivals A_n. As a consequence, if state w has not been already visited, we add it in $\mathcal{W}$. The state space grows until we have visited all the possible job arrival configurations.

To implement this modification, Algorithm 9 is changed at Line 4, where we replace $\mathcal{W}$ by $\mathcal{W}_n$, the set of states that have been visited up to sample n. Furthermore, we have to add an **if** condition inside the loop beginning at Line 7. This **if** condition checks whether the state is already present in the state space; otherwise, we add it in $\mathcal{W}_n$.

6.9.2 Performance Criteria

The first important point on which we focus is the evaluation of the performance of our learning algorithm. This is why we present in this section several criterion to evaluate both the learning performance and convergence of Algorithm 9.

1. To begin, we have to check if the probability transition matrix obtained after the learning phase of (PL) represents well the "actual" transition probability matrix of the MDP. If this one is known, we can analyze its convergence. One criteria will be the norm of the difference between the two probability transition matrices, $\|\hat{P}_n - P\|$, versus the length n of the training phase. Even though in Section 6.6 we have an upper bound on the number of iterations of the learning phase to ensure a "good" convergence of the probability transition matrix (and also the resulting speed policy), during the simulation we can have in practice a smaller convergence iteration value, where we know the value of the probability transition matrix.

2. The other metric to analyze the learning performance is to compare our speeds policy obtained after (PL) with (OA), the optimal policy when we have zero information for future jobs. Therefore, we will compute in the following part the overhead of the consumption of (OA) in comparison with the speed policy obtained after a (PL) learning phase with a dynamic programming on the learned probability transition matrix. The overhead is defined as follows:

$$\mathit{overhead}_{(\mathrm{OA})vs(\mathrm{PL})} = \frac{v^{(\mathrm{OA})} - v^{(\mathrm{PL})}}{v^{(\mathrm{PL})}} \tag{6.20}$$

These two metrics will be used in the next section to evaluate (PL).

6.9.3 Simulation Results

To evaluate the performance of our learning algorithm (PL), we have simulated 70 different sets of job types. Each set consists of a number k of job types of the form $J_i = (p_i, c_i, d_i)$, such that:

- p_i is the arrival probability of J_i, randomly chosen in the interval $[0.1, 0.9]$. This arrival model means that, at each time instant, a job of type J_i has a probability p_i to be released, for each $i \in \{1, .., k\}$. We can thus have, at a given time instant, between 0 and k jobs that are released simultaneously.
- c_i is the size of J_i, randomly chosen in the set $\{0, .., 4\}$.
- d_i is the deadline of J_i, randomly chosen in the set $\{1, 2\}$.
- $k \leq 5$ and $\sum_{i=1}^{k} p_i \in [0.8, 1]$. These two choices ensure that the generated systems are "interesting" for the learning phase, *i.e.*, they have an average load such that the range of feasible speeds in not reduced to a singleton (as it would be the case, for instance, if at each instant the cumulated size of job arrivals would amount to a load equal to $s_{\max}$).

Moreover, the chosen value of the discount γ is 0.9. Regarding $R_{\max}$, is differs for each set of jobs and it is bounded by $s_{\max}^3$. Since we have a discount factor $\gamma < 1$, (PL) uses the following set of speeds $\{s^{(\mathrm{OA})}(w) - 1, s^{(\mathrm{OA})}(w), s^{(\mathrm{OA})}(w) + 1, s^{(\mathrm{OA})}(w) + 2\}$. The additional speed in this set, $s^{(\mathrm{OA})}(w) - 1$, is due to the fact that we are in the discounted case (see Section 7.5 of Chapter 7). Recall that the choice of the discount case (and not the undiscounted) for the simulation is made to be able to compare the result of this chapter with the results of Chapter 7.

In order to understand more precisely the distribution of the different random simulations, we use in the upcoming figures a box representation, where each box is composed of:

- A black central line represents the median value.
- The upper bound and lower bound of the rectangle represent respectively the 3^{rd} and 1^{st} quartile.
- The end of the segment is 1.5 times the inter-quartile distance.
- The possible white bullets are the extremal value that are not included in the previous cases.
- The red line and the red points represent the evolution and the value of the mean of the set of set of jobs.

Fig. 6.2 displays the evolution of the difference $\|\hat{P}_n - P\|$, where $\hat{P}_n$ is the transition probability matrix learned after n iterations, n being the length of the training period, ranging between 10^3 and 10^7.

Suppose now that the user wishes the energy error in the discounted case, $\|\hat{V}_n^{\gamma} - V^{*,\gamma}\|$, to be less than 10^{-2}. Recall that Eq. (6.12) relates the energy performance and the probability matrix difference:

$$\|\hat{V}_n^{\gamma} - V^{*,\gamma}\| \leq \frac{R_{\max}}{1-\gamma} \|\hat{P}_n(s^*) - P(s^*)\|$$

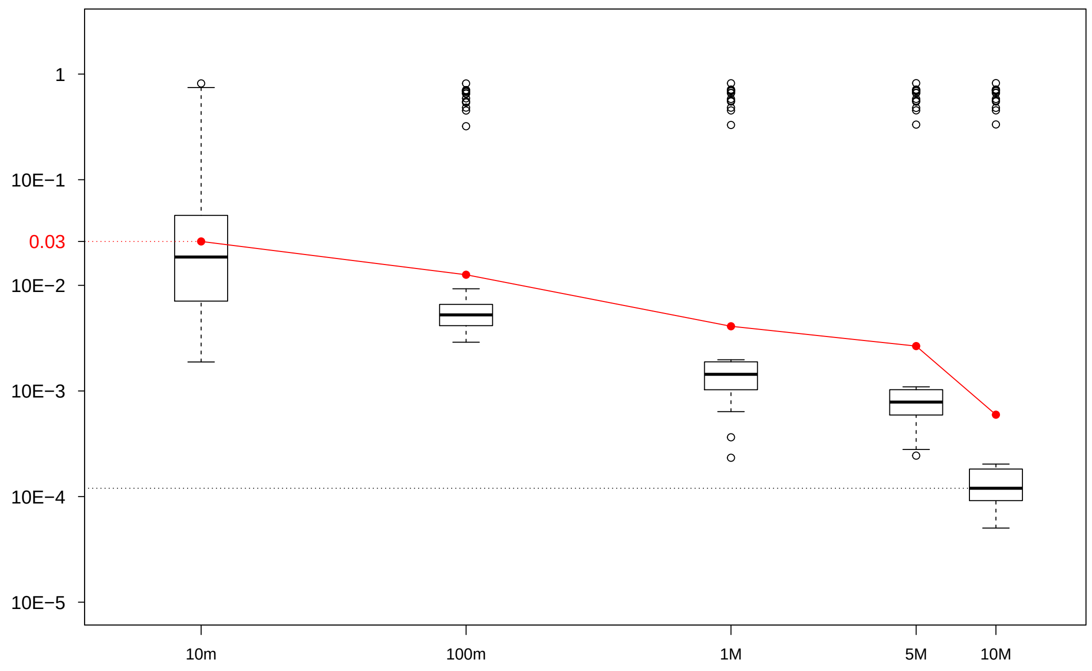


Figure 6.2.: Evolution of the norm $\|\hat{P}_n - P\|$ in function of the duration n of the learning phase: $n \in \{10^3, 10^4, 10^5, 10^6, 5.10^6, 10^7\}$.

where $R_{\max} = P_{ower}(s_{\max})$. Here, $s_{\max}$ differs for each set of jobs for feasibility reason. Indeed, (PL) uses the set of speeds $\{s^{(\mathrm{OA})}(w) - 1, s^{(\mathrm{OA})}(w), s^{(\mathrm{OA})}(w) + 1, s^{(\mathrm{OA})}(w) + 2\}$, hence $s^{(\mathrm{PL})}_{\max} = s^{(\mathrm{OA})}_{\max} + 2$. As we will prove in Chapter 8 (more precisely, in Theorem 8.1), the feasibility condition for (OA) is:

$$s^{(\mathrm{OA})}_{\max} \geq C\,(h_{\Delta-1} + 1)$$

where C is the maximum amount of work that can arrive at any instant (*i.e.*, $C = \sum_{i=1}^{k} c_i$), Δ is the maximum relative deadline (*i.e.*, $\Delta = \max_{i=1}^{k} d_i$), and h_n is the n-th harmonic number (*i.e.*, $h_n = \sum_{i=1}^{n} 1/i$).

This yields the following inequality:

$$\|\hat{V}^{\gamma}_n - V^{*,\gamma}\| \leq \|\hat{P}_n - P\| \frac{(C(h_{\Delta-1} + 1) + 2)^3}{1 - \gamma} \tag{6.21}$$

Consider for example a simulation where $C = 4$ and $\Delta = 2$. We then have $C(h_{\Delta-1} + 1) + 2 = 10$, hence $\frac{(C(h_{\Delta-1}+1)+2)^3}{1-\gamma} = 10^4$. Eq. (6.21) therefore implies that it suffices to take $\|\hat{P}_n - P\| \leq 10^{-6}$ to ensure that $\|\hat{V}^{\gamma}_n - V^{*,\gamma}\| \leq 10^{-2}$. In Fig. 6.2, we see that this *theoretical bound* on $\|\hat{P}_n - P\|$ is not yet reached after 10^7 learning steps. This can be seen as a poor performance, but in fact, as shown in Fig. 6.3, the convergence is achieved *in practice* after only 10^4 learning steps.

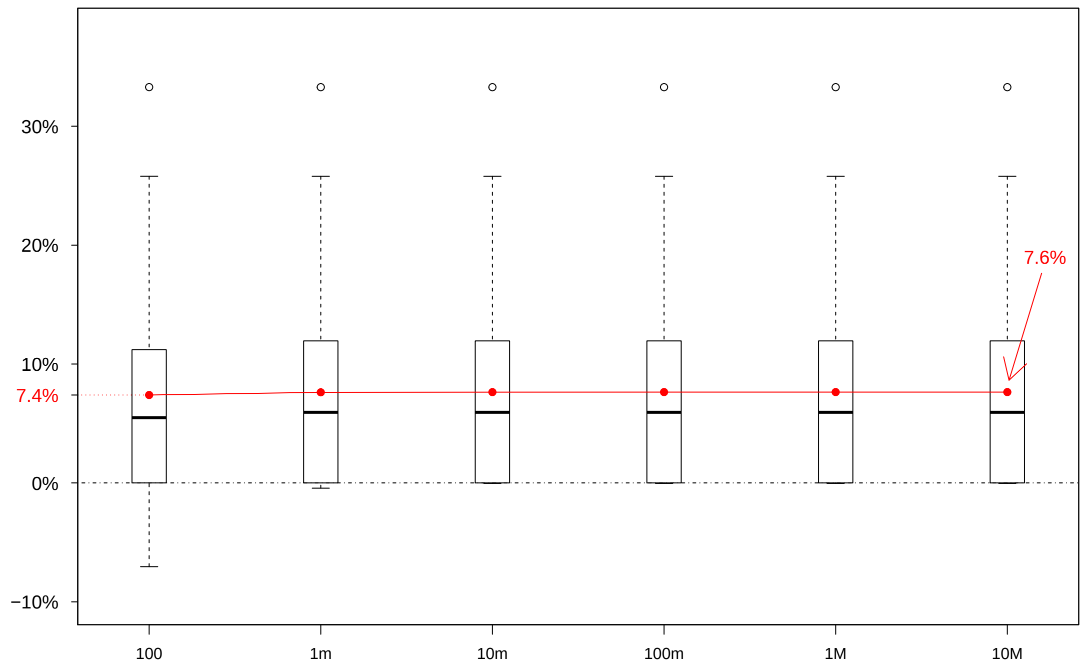


Figure 6.3.: Overhead in percentage of (OA) versus the learning algorithm (PL) depending on the duration n of the learning phase: $n \in \{10^2, 10^3, 10^4, 10^5, 10^6, 10^7\}$.

Fig. 6.3 depicts the overhead of the energy consumption of (OA) compared with the speed policy obtained with a dynamic programming on the probability transition matrix learned with (PL), in function of the duration n of the learning phase. As can be seen, the convergence for this second criterion is very fast: the speed policy learned with (PL) outperforms (OA) by 7.4% on average after 100 learning steps only and this percentages only progresses marginally when n increases (7.6% on average after 10^7 learning steps). These very good performances contrast significantly with the slow convergence rate on the first criterion, as depicted in Fig. 6.2.

Between 10^4 and 10^7 learning steps, all the box-plots have almost the same distribution: their mean value is 7.6%, their median value is 5.9%, and their quartiles are also identical. For $n = 10^2$, the results are slightly different: the mean value is 7.4% and the median is smaller to, at 5.5%. Once $n \geq 10^3$, the mean and median values remain constant. However there are some job sets that have a negative overhead. This means that, for these particular job sets, (OA) is better than the speed policy learned with (PL). Finally, once $n \geq 10^4$, all the overheads are strictly positive, meaning that for each set of jobs, the speed policy learned with (PL) outperforms (OA). This is very good result because 10^4 iterations is a reasonably small number for a learning phase.

It is interesting to compare Fig. 6.3 and Fig. 6.2: even though the convergence is not reached after 10^7 learning steps for (PL), the energy consumption of the speed policy learned with (PL) is always strictly better than that of (OA) after only 10^4 learning steps.

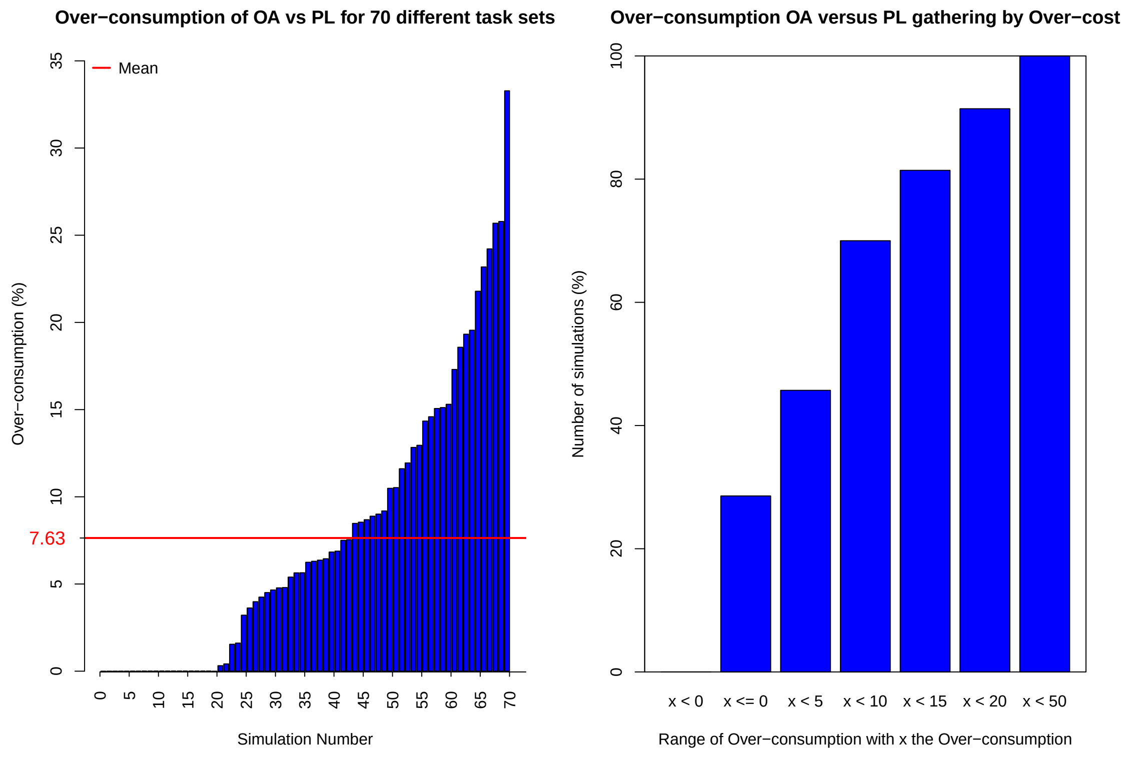


Figure 6.4.: Energy consumption overhead in percentage of (OA) versus the speed policy learned with (PL) after 10^3 learning steps. The job characteristics (p_i, c_i, d_i) are such that $p_i \in [0.1, 0.9]$, $c_i \in \{0, \ldots, 4\}$, and $d_i \in \{1, 2\}$.

Our final results are shown in Fig. 6.4, which depicts the energy consumption overhead in percentage of (OA) versus the speed policy learned with (PL) after 10^3 learning steps. The job characteristics $J_i = (p_i, c_i, d_i)$ are chosen as follows: the arrival probabilities p_i are randomly chosen in the interval $[0.1, 0.9]$, the sizes c_i are randomly chosen in the set $\{0, \ldots, 4\}$; and the deadlines d_i are randomly chosen in the set $\{1, 2\}$.

Fig. 6.4-left shows the energy overhead of each individual job set sorted by the resulting energy consumption overhead. (OA) outperforms (PL) over only one job set. Otherwise (PL) systematically outperforms (OA), and the average gain is 7.63%.

Fig. 6.4-right gathers the simulations by their over-consumption percentage value. Each vertical bar corresponds to an interval of energy consumption overhead of (OA) versus (PL). For instance, there are 46% job sets for which this energy overhead is below 5%.

6.10 Conclusion

This chapter shows that, with no statistical information on jobs, learning the transition probability matrix of the MDP can be a good opportunity to converge towards the optimal speed policy both cases, discounted and undiscounted.

The theoretical convergence criterion is long to be obtained, and yet the optimal speed policy is obtained after a short training period. Overall, the learned speed policy outperforms algorithms that do not take advantage of a learning phase, such as (OA).

Here, we have focused on synchronous learning. One extension will be to investigate also asynchronous learning, in order to minimize the undiscounted case. This has been studied in the literature and there are based on different technique than the one studied in this chapter. In Chapter 7 we will study an another method based on Q-learning to solve this problem, however this solution will be practical only for the discounted case, in contrary to this chapter where the two cases are possible.

7 Online Minimization: Learning the Energy Consumption with Unknown Statistics

7.1 Introduction

In Chapter 6 we proposed an algorithm to learn the Markov Decision Process (MDP) parameters, and from this knowledge we showed how to compute the online speed policy that minimizes the *undiscounted average energy consumption* by solving the MDP with a classical dynamic programming algorithm (in our case Value Iteration). In this chapter we focus on a different problem: We want to minimize the *discounted total energy consumption* of the processor. Intuitively, having a discount factor γ strictly less than 1 means that a given speed has a higher impact on the energy consumption *now* than the same speed used *in the past*. The "real world" justification for a discount is to consider that the electricity price is subject to inflation, the rate of which is exactly the inverse of the discount factor γ.

To achieve this, we propose in this chapter to learn directly the energy consumption of the optimal policy, and therefore the optimal speed policy, without knowing the MDP parameters as we did in Chapter 6.

Since we want to learn a policy that chooses the processor speed in order to maximize the cumulative energy cost of the chosen speeds by using the results of previous speed choices, the learning part should focus on *reinforcement learning* techniques. Furthermore, since we have no idea on the arrival job model and on the job characteristics, we are in the field of *model-free reinforcement learning*, with a discrete state space (*i.e.*, all the job characteristics, although unknown, are assumed to be in $\mathbb{N}$) and a discrete action space (*i.e.*, all the speeds used will be in $\mathbb{N}$).

One solution to solve such a reinforcement learning problem is to use *temporal difference methods*, which generalize the *Value Iteration* algorithm presented in Chapter 4. It is based on Bellman's equation and Sutton has proposed the first algorithms in [SB98]. One of the temporal difference algorithm is the Q-learning algorithm [WD92].

Our analysis is consists of two parts:

- The first part involves learning the optimal speed policy, in order to obtain the best Q-matrix value.
- The second part involves comparing the learning cost of the learned speed policy with that of (OA) (see Chapter 4).

Let us first give an informal description of the behavior of the system as well as the behavior of the learner that selects the processor speed at each time.

Jobs arrive during the execution of the system, and at each instant the speed policy must choose one available speed. More specifically, after n samples, the processor use an available speed s_n to execute partially or totally jobs that are present in the system at that time.

The total energy cost when starting in state w at time 0 is the *discounted energy* spent over time, with discount factor $\gamma < 1$:

$$V(w) = \sum_{n=0}^{\infty} \gamma^n P_{\mathit{ower}}(s_n). \tag{7.1}$$

As said in Section 7.1, the discount factor puts more emphasis on the “recent” choices of processor speed than on the “ancient” ones.

According to these notations, our MDP will denoted $(P_{\mathit{ower}}, \mathcal{W}, P, \gamma)$ in the following.

7.2 Problem Statement

In this chapter, we will determine the optimal speed policy that minimize the total energy consumption. As said in Section 7.1, the problem is different from Chapter 6 because we want to minimize the *total energy consumption* (and not the average), and furthermore we consider that there is a *discount factor* $\gamma < 1$. With these problem characteristics, we have to use a different method than this one used in Chapter 6 to determine the optimal speed policy.

We do not know the job arrival distribution, so we have no information to determine the speed policy. Moreover, we do not know the MDP parameters (see Section 6.2 of Chapter 6). Instead of discovering the MDP parameters by a learning method, we learn directly the cost of the different speed policies, which then allows us to determine the optimal speed policy. To do that we will the well known Q-learning algorithm.

As in Chapter 6, the new incoming information at each sample n is the *workload arrival function* A_n, where a workload accounts for zero, one or several jobs with the same arrival date. One learning step (*i.e.*, one iteration of the Q-learning algorithm) consists in getting a new sample of the A_n function. With this new information, the Q-learning algorithm updates the value knowledge and improves the cost of the speed policy.

There are two different Q-learning algorithm versions:

- One version is called *asynchronous* (AQL). It is the most “natural” version, in the sense that at each sample n with workload arrival A_n, we update the policy cost value *for the current state only*. At each sample, the learning of the policy cost value improves, and after a given horizon, we have converged towards the optimal policy. Section 7.3 studies (AQL) in details and provides a proof of convergence.

- Another version is called *synchronous* (SQL). In this version, at each n with workload arrival A_n, we update the policy cost value *for all the possible states* of the state space $\mathcal{W}$. The convergence is reached faster than with (AQL) more states and more speed choices are explored at each sample.

Both algorithms (AQL) and (SQL) can be used in two different cases. Either in two phases as in Chapter 6, first a *learning phase* during which the speed policy is learned with Q-learning but the energy is not measured, and then an *application phase* during which the speed policy that has been learned is used and the energy is measured. Or with a *combined phase* during which the speed policy is learned with Q-learning and used, and of course the energy is measured too.

The following section will be devoted to the description of the Q-learning Algorithm.

7.3 Q-learning Algorithm

7.3.1 Synchronous Q-learning Algorithm (SQL)

Consider the discounted MDP $(P_{ower}, \mathcal{W}, P, \gamma)$, as described in Chapter 6.

The goal of this section is to show how a learner (selecting the processor speed) can choose the speed s_n at sample n so that it eventually converges to the best possible choices while the job arrival probabilities are not known (*i.e.*, $P(w, s, w')$ is not known). Instead we assume that a *trajectory* (*i.e.*, an infinite sequence of random jobs), distributed according to the probability distribution of the *past* job arrivals, is provided to the learner, one arrival at a time. In the following, this sequence of jobs will be represented by its workload trajectory $(A_n)_{n\in\mathbb{N}}$.

By definition, the Q-value function $q^*(w, s)$ is the *minimal discounted energy consumption* starting in state w at time 0, using speed s at the first time step. The optimal cost V^* is related to q^* as follows:

$$V^*(w) = \min_s q^*(w, s). \tag{7.2}$$

The Bellman Optimality Equation (BOE) for the Q-values is:

$$\begin{aligned} q^*(w,s) &= P_{ower}(s) + \gamma \sum_{w'} P(w,s,w') V^*(w') && (7.3) \\ &= \sum_{w'\in\mathcal{W}} P(w,s,w') \left[P_{ower}(s) + \gamma \min_{s'} q^*(w',s') \right]. && (7.4) \end{aligned}$$

Let us now introduce the operator F from the set of Q-value functions to itself:

$$F(q)(w,s) = \sum_{w'\in\mathcal{W}} P(w,s,w') \left[P_{ower}(s) + \gamma \min_{s'} q(w',s') - q(w,s) \right]. \tag{7.5}$$

In functional form, Eq. (7.4) says that q^* can be seen as a *root* of operator F: $F(q^*) = 0$.

The value iteration for Q-values is $\hat{q}_{n+1} = \hat{q}_n + F(\hat{q}_n)$ and, because F is γ-contracting, it converges towards the optimal q^*, starting from an arbitrary value for $\hat{q}_0$.

The Synchronous Q-learning algorithm (SQL) (Algorithm 10) replaces the expected cost-to-go term in Eq. (7.4) (*i.e.*, the right term of the addition) by a realisation (*i.e.*, the $\min_{s'} \hat{Q}_n(W, s')$ term) involving the next random event (job arrival A_n) and computes a sequence of random variables Q_n that mimics Eq. (7.4), using a vanishing sequence of learning factors, $(\lambda_n)_{n\in\mathbb{N}}$, that converges towards 0 when $n \to \infty$.

Algorithm 10: Synchronous Q-learning Algorithm (SQL)

```
1: $\hat{Q}_0(w, s)$ is set to an arbitrary value (e.g., 0) for all $w$ and $s$;
2: while $n < N$ do
3:     for all state $w$ and all admissible speed $s$ do
4:         Get the new incoming jobs $A_{n+1}$
5:         $W \leftarrow Succ(w, s, A_{n+1})$;
6:         $\hat{Q}_{n+1}(w, s) \leftarrow (1 - \lambda_n)\hat{Q}_n(w, s) + \lambda_n(P_{ower}(s) + \gamma \min_{s'} \hat{Q}_n(W, s'))$;
7:     end for
8: end while
```

recall that N is the total number of samples. Line 4 is typeset in italic blue to insist on the fact that this part of Algorithm 10 depends only on the system behavior. It corresponds to the fact that we observe the characteristics of the jobs that arrive at each time steps.

The convergence of the Q-learning algorithm was proved for the first time in [WD92]. Here we provide a simple proof of convergence for the synchronous version of the Q-learning algorithm, based on *stochastic approximation theory*.

Theorem 7.1. *The random variables $\hat{Q}_n(w, s)$ computed by* (SQL) *converge almost surely to the optimal Q-values $q^*(w, s)$, and hence,* (SQL) *asymptotically learns the optimal speed policy.*

Proof. First, one can easily check that the random variables $\hat{Q}_n(w, s)$ are bounded by $P_{ower}(s_{\max})/(1-\gamma)$ for any n, w, and s.

Now, to show that $\hat{Q}_n(w, s) \to q^*(w, s)$ asymptotically, let us first rewrite the evolution of $\hat{Q}_n$ given in line. (6) of Algorithm 10 as:

$$\hat{Q}_{n+1}(w, s) = \hat{Q}_n(w, s) + \lambda_n \left[P_{ower}(s) + \gamma \min_b \hat{Q}_n(W, b) - \hat{Q}_n(w, s) \right]. \tag{7.6}$$

Eq. (7.6) can be interpreted as a stochastic approximation of the following deterministic Ordinary Differential Equation (ODE):

$$\dot{q}(w, s) \quad = \quad \mathbb{E}_W \left[P_{ower}(s) + \gamma \min_b q(W, b) - q(w, s) \middle| (w, s) \right]. \tag{7.7}$$

In a compact form, and using the operator F introduced in Eq. (7.5), this ODE is written as:

$$\dot{q} = F(q). \tag{7.8}$$

On the one hand, the operator F is γ-contracting, therefore the ODE (7.8) has a unique global attractor, denoted q^∞. The theory of stochastic approximation says that, if the sequence λ_n is such that $\sum \lambda_n$ diverges and $\sum \lambda_n^2$ converges (this double condition is known as "$L2 - L1$"[1]), then the bounded sequence $\hat{Q}_n$ converges towards the unique solution q^∞ of the ODE, almost surely [BM00]. Notice that the unique asymptotic attractor q^∞ necessarily satisfies the following equation:

$$F(q^\infty) = \mathbb{E}_W \left[P_{ower}(s) + \gamma \min_b q^\infty(W, b) - q^\infty(w, s) | (w, s) \right] = 0. \tag{7.9}$$

On the other hand, the value iteration equation to compute the optimal Q-values can be written as an *expectation* (see Eq. (7.4)):

$$\begin{aligned} q_{n+1}(w,s) &= \sum_{w'} P(w,s,w') \left[P_{ower}(s) + \beta \min_b q_n(w', b) \right] & (7.10)\\ &= q_n(w,s) + \sum_{w'} P(w,s,w') \left[P_{ower}(s) + \gamma \min_b q_n(w', b) - q_n(w,s) \right] & (7.11)\\ &= q_n(w,s) + \mathbb{E}_W \left[P_{ower}(s) + \gamma \min_b q_n(W, b) - q_n(w,s) \big| (w,s) \right]. & (7.12) \end{aligned}$$

By inspecting (7.12), one can see directly that its fixed point, namely the optimal Q-value q^*, satisfies $F(q^*) = 0$, and hence q^* is equal to q^∞. □

After proving that (SQL) converges, let us present in the next section the asynchronous case with the (AQL) algorithm.

7.3.2 Asynchronous Q-learning Algorithm (AQL)

The asynchronous version of the Q-learning Algorithm is different from the synchronous one. It only updates the Q-value of the current state, along the workload trajectory $(A_n)_{n \in \mathbb{N}}$.

Algorithm 11: Asynchronous Q-learning Algorithm (AQL)

```
1: $\hat{Q}(w, s)$ is set to an arbitrary value (e.g., 0) for all $w$ and $s$;
2: $w_0$ is the initial state;
3: while $n < N$ do
4:    Select speed $S_n \leftarrow h_n(W_n)$;
5:    Get the new incoming jobs $A_{n+1}$
6:    $W_{n+1} \leftarrow Succ(W_n, S_n, A_{n+1})$;
7:    $\hat{Q}(W_n, S_n) \leftarrow (1 - \lambda)\hat{Q}(W_n, S_n) + \lambda(P_{ower}(S_n) + \gamma \min_{s'} \hat{Q}(W_{n+1}, s'))$;
8: end while
```

[1] X_n converge towards X for the L^1 norm if $\lim_{n\to\infty} \mathbb{E}(| X_n - X |) = 0$, while X_n converge towards X for the L^2 norm if $\lim_{n\to\infty} \mathbb{E}(| X_n - X |^2) = 0$.

The learning factor λ now depends on the number of visits to the state w, denoted $Nb_{visit}(w)$. Its value is chosen as follows in our numerical tests, although any value satisfying the $L2 - L1$ constraint would work:

$$\lambda(w) = \frac{1}{Nb_{visit}(w)^{2/3}}$$

Theorem 7.2. *Convergence of Q-learning algorithm [WD92]. If the selection function h_n is such that each pair (w,s) is visited an infinite number of times during the execution of the system, then each Q-value converges towards its optimal value almost surely.*

Proof. (Sketch) The proof is the similar to the proof for (SQL) by adding asynchronous convergence arguments (see for example [BT96]). □

The selection function h_n chosen in our numerical experiments is typically *ε-greedy* or *softmax* as presented in [SB98]. The choice between them depends on the performance metrics. Here, *softmax* is often used because it seems to perform better (see the experimental part: Section 7.7). Some others algorithms, such as [Aza+11] and [DM17], have been presented in the literature and are based on the same methods. In this chapter, we will focus in particular on the Q-learning algorithm.

7.4 Structural Properties of the Q-learning Algorithm

7.4.1 Synchronous Q-learning Algorithm (SQL)

In this section, we show that (SQL) offers several monotonicity properties.

Lemma 7.1. *By setting $\hat{Q}_0(w,s) = \frac{P_{ower}(s_{\max})}{1-\gamma}$, then for all n, $\hat{Q}_{n+1}(w,s) \leq \hat{Q}_n(w,s)$ for all (w,s).*

Proof. The proof holds by induction. Case $n = 1$:

$$\begin{aligned}
\hat{Q}_1(w,s) &= (1-\lambda_1)\hat{Q}_0(w,s) + \lambda_1 \left(P_{ower}(s) + \gamma \min_{s' \in \mathcal{S}} \hat{Q}_0(W', s') \right) \\
&= (1-\lambda_1) \sum_{i=0}^{\infty} \gamma^i P_{ower}(s_{\max}) + \lambda_1 \left(P_{ower}(s) + \gamma \sum_{i=0}^{\infty} \gamma^i P_{ower}(s_{\max}) \right) \\
&\leq (1-\lambda_1) \sum_{i=0}^{\infty} \gamma^i P_{ower}(s_{\max}) + \lambda_1 \left(\sum_{i=0}^{\infty} \gamma^i P_{ower}(s_{\max}) \right) \qquad (7.13) \\
&= \hat{Q}_0(w,s).
\end{aligned}$$

Inequality (7.13) follows from the fact that the function $P_{ower}(s)$ is non-decreasing. The general case follows, $\forall(w,s)$, by simple inspection of line 6 of Algorithm 10:

$$\begin{aligned}\hat{Q}_{n+1}(w,s) &= (1-\lambda_n)\hat{Q}_n(w,s)+\lambda_n\left(P_{ower}(s)+\gamma\min_{s'\in\mathcal{S}}\hat{Q}_n(W',s')\right) &(7.14)\\ &\le (1-\lambda_n)\hat{Q}_{n-1}(w,s)+\lambda_n\left(P_{ower}(s)+\gamma\min_{s'\in\mathcal{S}}\hat{Q}_{n-1}(W',s')\right) &(7.15)\\ &= \hat{Q}_n(w,s).\end{aligned}$$

Inequality (7.15) comes from the induction assumption and the monotonicity of all the operations involved in Eq. (7.15). □

Let $\hat{Q}^k$ be the Q-values obtained by (SQL) when the only speeds available in state w are in the set

$$\mathcal{S}^k=\left\{s^{(\mathrm{OA})}(w)-1,s^{(\mathrm{OA})}(w),s^{(\mathrm{OA})}(w)+1,\ldots,s^{(\mathrm{OA})}(w)+k-1\wedge s_{\max}\right\}. \qquad (7.16)$$

The corresponding version of (SQL) is denoted $(\mathrm{SQL})(k)$ in the following. We can state the following result.

Lemma 7.2. *By setting $\hat{Q}_0^k(w,s)=\hat{Q}_0(w,s)=\frac{P_{ower}(s_{\max})}{1-\gamma}$, then for all n and $k>0$, $Q^{(\mathrm{OA})}(w,s)\ge\hat{Q}_n^k(w,s)\ge\hat{Q}_n^{k+1}(w,s)\ge\hat{Q}_n(w,s)$ for all w,s.*

Proof. By induction. Case $n=0$: For all k, $\hat{Q}_0^k$ values are equal, so nothing needs to be proved. The general case n is proved by simple inspection of line (6) of Algorithm 10: $\forall(w,s)$,

$$\begin{aligned}\hat{Q}_n^{k+1}(w,s) &= (1-\lambda_{n-1})\hat{Q}_{n-1}^{k+1}(w,s)+\lambda_{n-1}\left(P_{ower}(s)+\gamma\min_{s'\in\mathcal{S}^{k+1}}\hat{Q}_{n-1}^{k+1}(W',s')\right)\\ &\le (1-\lambda_{n-1})\hat{Q}_{n-1}^{k}(w,s)+\lambda_{n-1}\left(P_{ower}(s)+\gamma\min_{s'\in\mathcal{S}^{k+1}}\hat{Q}_{n-1}^{k}(W',s')\right) &(7.17)\\ &\le (1-\lambda_{n-1})\hat{Q}_{n-1}^{k}(w,s)+\lambda_{n-1}\left(P_{ower}(s)+\gamma\min_{s'\in\mathcal{S}^{k}}\hat{Q}_{n-1}^{k}(W',s')\right) &(7.18)\\ &= \hat{Q}_n^k(w,s).\end{aligned}$$

Inequality (7.17) comes from the induction assumption and Inequality (7.18) comes from the fact that $\mathcal{S}^k\subset\mathcal{S}^{k+1}$. □

7.4.2 Asynchronous Q-learning Algorithm (AQL)

The monotony properties of the (SQL) do not extend to (AQL). Monotonicity in n does not hold because the sequence of states w_n is a random sequence of states that cannot be compared. As for the monotonicity in k, it does not hold because the number of visits to a given state (w,s) depends on k, and so its Q-value may be large for some large values of k if the state (w,s) has not been visited often under $\hat{Q}^k$, while it could be small for a smaller value of k' if it has been visited often under $\hat{Q}^{k'}$.

7.5 Relevant Speed Analysis

In Chapter 6, we proved that in the undiscounted case, the optimal processor speed is always above $s^{(\mathrm{OA})}(w)$, and so taking into account only speeds larger than $s^{(\mathrm{OA})}(w)$ can decrease the learning speed set. However, since we consider here a discount factor $\gamma < 1$, we have to re-evaluate this bound.

The goal of this section is to study the impact of the discount factor on the optimal speed policy for one job $J(C, \Delta)$ when the power function is a cubic function. For the undiscounted case, since the power function is convex, the optimal speed schedule to execute this job is to run the processor at a speed equal to $\frac{C}{\Delta}$ at each time step between 0 and $\Delta - 1$.

The goal is to see how the optimal speed policy for the undiscounted case is far from the computed optimal speed policy in the discounted case. To begin, let us compute the optimal speed policy in the discounted case. This means we have to solve the following system:

$$\min_{s_1,..,s_\Delta} \left\{ \sum_{i=1}^{\Delta} \gamma^{i-1} s_i^3 \right\} \text{ under constraints } \sum_{i=0}^{\Delta-1} s_i = C. \tag{7.19}$$

The Lagrangian of the System (7.19) is:

$$L(s_i, \lambda) = \sum_{i=1}^{\Delta} \gamma^{i-1} s_i^3 - \lambda \left(\sum_{i=1}^{\Delta} s_i - C \right). \tag{7.20}$$

After derivation, we have:

$$s_1^2 = \gamma\, s_2^2 = \ldots = \gamma^{\Delta-1}\, s_\Delta^2. \tag{7.21}$$

By letting $d = \Delta - 1$, the max is therefore obtained for:

$$\begin{aligned} s_1 &= \gamma^{\frac{d}{2}}\, s_\Delta && (7.22)\\ s_2 &= \gamma^{\frac{d-1}{2}}\, s_\Delta && (7.23)\\ &\vdots \\ s_{\Delta-1} &= \gamma^{\frac{d-(\Delta-2)}{2}}\, s_\Delta. && (7.24) \end{aligned}$$

By replacing these values in Eq. (7.19), we obtain the speed s_Δ:

$$s_\Delta = C \left(\sum_{i=0}^{d} \gamma^{\frac{i}{2}} \right)^{-1}. \tag{7.25}$$

By replacing in (7.19) with Eqs. (7.22) to (7.24), the general energy cost of the speed policy is:

$$\text{CostWithOptimalPolicy} = C^3\gamma^d \left(\sum_{i=0}^{d} \gamma^{\frac{i}{2}}\right)^{-2}. \tag{7.26}$$

As we said previously, for job J, if there is no discount factor, by convexity of the power function, the best choice for the energy consumption is to use the speed $\frac{C}{\Delta}$ all time instants. Thus the energy cost of this speed policy in the discounted case is:

$$\text{CostWithNoDiscountPolicy} = \left(\frac{C}{\Delta}\right)^3 \sum_{i=0}^{d} \gamma^i. \tag{7.27}$$

Therefore, the over-consumption of the undiscounted speed policy is :

$$\frac{\text{CostWithNoDiscountPolicy}}{\text{CostWithOptimalPolicy}} = \frac{1}{\Delta^3\gamma^d}\left(\frac{1-\gamma^{\frac{d}{2}}}{1-\gamma^{\frac{1}{2}}}\right)^3 \frac{1+\gamma^{\frac{d}{2}}}{1+\gamma^{\frac{1}{2}}}. \tag{7.28}$$

Now we want to find an equivalent of Eq. (7.28) when we are getting closer to the undiscounted case, *i.e.*, when $\gamma \to 1$. By setting $\gamma = 1-\varepsilon$, the limited development of Eq. (7.28) is:

$$\frac{\text{CostWithNoDiscountPolicy}}{\text{CostWithOptimalPolicy}} \sim 1 - \frac{3}{4}(d-2)\varepsilon = 1 - \frac{3}{4}(\Delta-3)\varepsilon \tag{7.29}$$

Eq. (7.29) shows that the over-consumption of the undiscounted policy is of $\frac{3}{4}(\Delta-3)(1-\gamma)$.

For the undiscounted case, we have noted in the previous section that considering only speeds faster or equal to $s^{(\mathrm{OA})}$ in Q-learning let us to obtain the optimal policy. Regarding the discounted case, in this section, we have shown that adding $s^{(\mathrm{OA})}-1$ in the set of available speeds is enough to counterbalance the effect of the discount factor γ. As a consequence, in the following, we will reduce the Q-learning speed choices for each state to the speeds faster than $s^{(\mathrm{OA})}-1$.

7.6 Feasibility of the Q-learning Algorithm

Section 7.5 shows us that $s^{(\mathrm{OA})}-1$ is the minimal speed that can be chosen by the Q-learning algorithm. As the minimal speed changes, we have also a different feasibility condition on the maximal speed processor $s_{\max}$, when we compared with Chapter 6.

Based on the proof developed in the undiscounted case (see Chapter 6), a necessary condition to ensure that the Q-learning Algorithm is feasible with a speed choice above $s^{(\mathrm{OA})}-1$ is:

$$s_{\max} \geq C\log(\Delta) + \Delta - 2 \tag{7.30}$$

To find this bound, we use the same technique as in the proof of Theorem 8.1 of Chapter 8, but by considering $s^{(\mathrm{OA})}-1$ as the minimal speed choice instead of $s^{(\mathrm{OA})}$.

7.7 Experimental Section

7.7.1 Use Cases

We apply our model-free reinforcement learning framework to two distinct case studies.

- **Training case study:** The first case study considers a training period during which the learning algorithm is used over one or several typical job sequences, in order to learn the optimal speed policy. Once the training period is over, the learned speed policy is used *in production* to save energy in the actual real-time system. The performance metric is the length of the training period and the quality of the learned speed policy. (SQL) is adapted to this case study. We perform several experiments showing the average performance gap of the policy computed by (SQL) over a training period that ranges from 10^3 to 10^7 samples, w.r.t. the optimal speed policy.

- **Online-cost case study:** The second use case does not involve a training period. The energy cost of the learning algorithm is accounted for from the start: The initial bad choices and their high energy cost cannot be dismissed. In this case, the only usable learning algorithm is (AQL), since speed choices have to be made online.

  In the online-cost case study, the performance metric will be the *regret* w.r.t. the optimal speed policy. The total cost being discounted, the definition of the regret must be adapted. To do so, we consider the discount factor as a stopping probability of the system at each step. The energy cost is cumulated up to the stopping time. The system is then restarted in its initial state and the same dynamic is taken over a new period (the Q-values learned in the first period are kept) until the system stops again, and so on and so forth. The energy cost is cumulated over N periods and the regret is

$$R_N^{(\mathrm{AQL})} = \sum_{n=1}^{N} \sum_{i=0}^{T_n} \left(P_{ower}(s_i^k) - P_{ower}(s_i^*) \right), \tag{7.31}$$

  where T_n is a random size of the n-th period (up to the n-th stopping time), s_i^k is the speed chosen by (AQL) at sample i, and s_i^* the speed chosen by the optimal policy under the same sequence of job arrivals.

  In the following, we perform several experiments showing the average regret w.r.t. the optimal speed policy of (AQL) for several time horizons. All the simulations are done with the following set of processor speed:

$$\mathcal{S}_{possible}(w_n) = \left\{ s^{(\mathrm{OA})} - 1, s^{(\mathrm{OA})}, s^{(\mathrm{OA})} + 1, s^{(\mathrm{OA})} + 2 \right\} \tag{7.32}$$

  where w_n is the current state at time instant n. The rationale behind this choice of available speeds is that (OA) is close to the optimal, as we have shown in Section 7.5, and so the optimal speeds are expected to be close to the speed chosen by (OA). Moreover, with this reduced choice we reduce the set of speeds, and hence the duration of our experiments. This choice concerns both (AQL) and (SQL).

Most of the experiments reported here belong to the first case, *i.e.* the training case study. The second case, *i.e.* the online-cost case, is an ongoing work, and so only one job set has been analyzed

7.7.2 State Representation in (SQL)

Since we have no knowledge on the maximal deadline Δ and the maximal size S of the jobs beforehand, the state space is not known in advance, so that one cannot use (SQL) Algorithm as is. In line 3 of the algorithm we replace $\mathcal{W}$ by $\mathcal{W}_n$, the set of states that have been visited up to sample n. In this respect, this new version is intermediate between the synchronous and asynchronous Q-learning Algorithm. If we also denote by $\mathcal{W}_n^k$ the set of states that (SQL) has visited at sample n, then, $\mathcal{W}_n^k$ is growing at sample n and at k, by construction. Therefore, the monotonicity property given in Lemma 7.1 is still true for this new version of the (SQL) algorithm with a growing state space.

As described in Chapter 6, the size of the state space can be huge, since it depends on the number of different jobs and on the maximal parameter values used during the algorithm. As a consequence, we use a hash-table to access the Q-matrix value in the Q-learning Algorithm.

7.7.3 Training Offline: (SQL) Evaluation

The offline training, done by (SQL) for n samples, leads to a specific Q-matrix $\hat{Q}_n$. With this Q-matrix, we can deduce a speed policy obtained by Q-learning after n samples, that attributes for each state a processor speed to apply. This speed policy is defined as follows:

$$s_n(w) = \operatorname*{argmin}_{s \in \mathcal{S}} \hat{Q}_n(w, s) \tag{7.33}$$

In this part, we want to evaluate the Q-learning speed policy s_n (SQL) by comparing it with the optimal policy, *i.e.*, Value Iteration (VI), and Optimal Available (OA). To determine the optimal policy, we have to know more information about the job stream: sizes, deadlines and arrival probabilities of each job met during the process (*i.e.*, as in Chapter 2). Let us suppose a known process, where we compute the optimal policy (it can be computed by value or policy iteration algorithm, see [Put05]). We compare this optimal policy with all (SQL) that are determined with no knowledge on the system.

We use two different comparisons to check how far is (SQL) from the optimal:

1. The first one is to evaluate **the performance of the speed policy** (SQL) **from the starting state** $w_0 = (0, 0)$. This case represents the performance of the policy if we restart the system from scratch.

2. The second one is to evaluate **the performance of the speed policy** (SQL) **for each possible starting state**. We assign a weight to each value, proportional to the time spent in each state under this policy. We compare these weighted sums for (VI) and each variant of (SQL).

This represents the performance of the policy if we continue the process from the last state of the training process.

To compute the performance $\hat{V}_n$, which is a vector of performance of states, we have to build the probability transition for each speed policy. If we know, for each state w, the speed policy $s_n(w)$ and the knowledge of job probabilities, we can determine the probability matrix P_n to go from one state to another at a fix speed policy s_n. The performance $\hat{V}_n$ is therefore given by:

$$\hat{V}_n = (\mathbb{I} - \gamma P_n)^{-1} P_{\text{ower}}(s_n) \tag{7.34}$$

Here we can determine the first comparison method 1, by comparing $\hat{V}_n(w_0)$ for (VI), and (SQL).

Furthermore, due to the knowledge of the transition probability matrix P_n, we can compute the stationary measure μ_n of the Markov chain determined by the matrix P_n. $\mu_n(w)$ is the proportion of time we spend in the state w under speed policy s_n and is computed as follows:

$$\mu_n P_n = \mu_n \tag{7.35}$$

Let us notice that μ_n is not a known parameter, but it can be measured in practice during the system execution.

Now, to obtain the second comparison method 2, we compute the weighted performance as follows:

$$\sum_{i=1}^{|\mathcal{W}|} \mu_n(i) \hat{V}_n(i) \tag{7.36}$$

Figure 7.1 represents the weighted performance for each policy (SQL) and (VI) for different sample number of the Q-learning algorithm (SQL). This simulation is realized with the parameters of the Ex. 7.1, and with a discount factor $\gamma = 0.99$. The learning factor λ is defined in Eq. (7.13).

Example 7.1. *Job* J_1 *is* $(p_1 = 0.4, c_1 = 4, d_1 = 3)$. *Job* J_2 *is* $(p_2 = 0.4, c_2 = 4, d_2 = 1)$. *The maximal processor speed is set to* 15. *The energy consumption per time unit is* $P_{\text{ower}}(s) = s^3$.

In Fig 7.1, (OA) policy consumes the most energy, with a mean energy of $6550J$. The optimal policy when we know the job statistics, (VI), consumes the less, with a mean energy of $4803J$. On this example, the convergence of the Q-learning algorithm towards the optimal policy (VI) is quick, in less than 1000 samples (iteration numbers) when we let 3 speeds choice. When there are only 2 speeds, the stability value of the mean energy is $5506J$, that leads to a lost of 14.6% in comparison with the optimal.

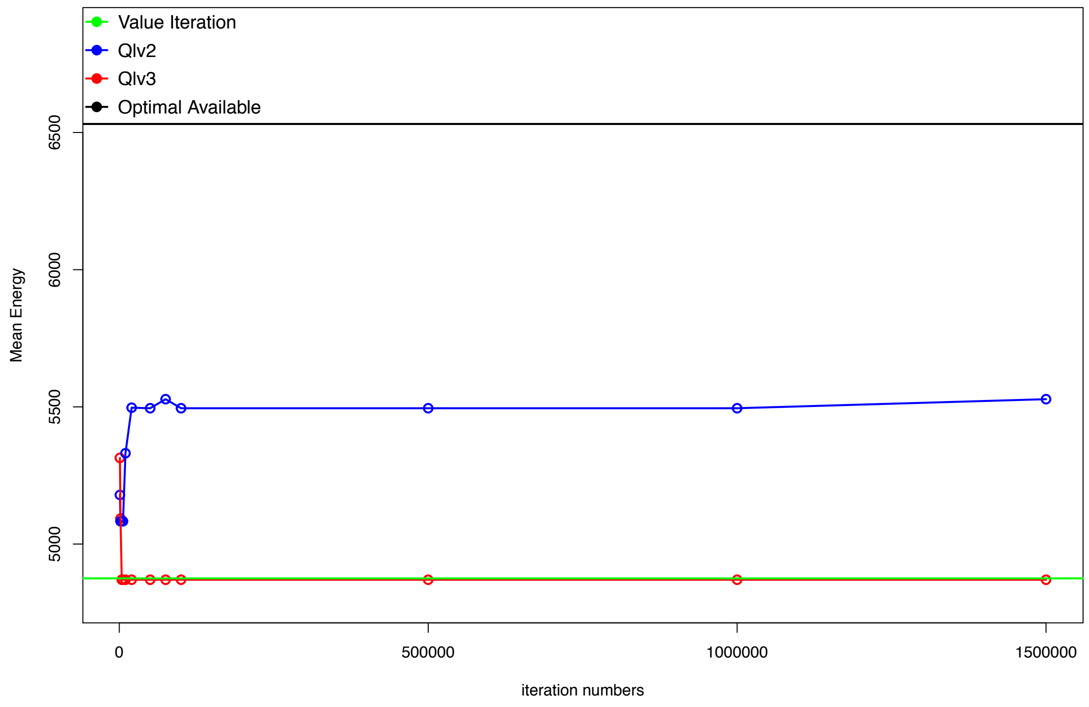


Figure 7.1.: Comparison of the cost of the policy weighted by the visit of each states for (SQL), (MDP) and (OA). The discount factor is $\gamma = 0.99$. There are 2 jobs $J_1(4,3)$ that comes 40% of the time and $J_2(4,1)$ 40% of the time also. During 20% of the time, no job is coming. The processor maximal speed is bounded by 15. Qlv2 is (SQL) with $s^{(\mathrm{OA})}$ and $s^{(\mathrm{OA})}+1$ speed choices and Qlv3 is (SQL) with $s^{(\mathrm{OA})}$, $s^{(\mathrm{OA})}+1$, and $s^{(\mathrm{OA})}+2$ speed choices

7.7.4 Simulation Results for (SQL)

In this section, we evaluate the performance of the Q-learning algorithm (SQL) after different learning periods. To compare the energy cost of each speed policy: (OA), (SQL) and the optimal policy (VI), we introduce two comparison indexes, Def. 7.1 and 7.2.

Definition 7.1. *Overhead of* (OA) *versus* (SQL)

$$\frac{V^{(\mathrm{OA})} - V^{(\mathrm{SQL})}}{V^{(\mathrm{SQL})}} \tag{7.37}$$

Definition 7.2. *Overhead of* (SQL) *versus the optimal policy*

$$\frac{V^{(\mathrm{SQL})} - V^{*}}{V^{*}} \tag{7.38}$$

The first experimentation done to test the performance of the Q-learning simulation is an analysis of one job stream. We first consider the job stream described in Example 7.1 of Section 7.7.3.

The second experimentation is a more general analysis on a set of Markov decision processes. Job numbers and jobs features, *i.e.* deadlines (bounded by $\Delta = 2$), sizes, and arrival time probabilities are generated randomly under an uniform law. The discount factor value is of $\gamma = 0.90$. The used set is the same as that used in Chapter 6.

The third experimentation is the same context as the second, but with a maximal relative deadline of 3 ($\Delta = 3$).

Experiment 1: A specific job stream

In this paragraph, we test the Q-learning algorithm on Ex. 7.1.

The Q-learning parameters are the discount factor, $\gamma = 0.90$, and the learning factor λ, defined in Eq. (7.13). In the following we will always consider these values for these parameters.

By doing that we note that after 2.10^3 learning steps, the *overhead* of (OA) over (SQL) (see Def. 7.1) is stabilized around 14%, more precisely 13.52% after 10^4 learning steps.

We can compare (SQL) versus the optimal policy because we know the system characteristics (here jobs definition). Actually, the jobs characteristics are unknown, in particular the arrival job probability. As a consequence, (SQL) is applicable, but the optimal speed policy (VI) can not be computed. The comparison versus (VI) is therefore virtual, and we use it purely to know how close we are to the optimal energy consumption value.

We have to note that the Q-matrix energy matrix and the processor speed choices have not yet converge, but the *overhead* is stabilized.

Experiment 2: Random family of job streams with Δ = 2

Other experimentations have been executed on a family of job streams to evaluate the performance of (SQL) algorithm. We use the same experimentation data than in the previous chapter, Chapter 6. We recall the structure of the system, which is composed of 70 different sets of job types. Each set consists of a number k of job types of the form $J_i = (p_i, c_i, d_i)$, such that:

- p_i is the arrival probability of J_i, randomly chosen in the interval $[0.1, 0.9]$. This arrival model means that, at each time instant, a job of type J_i has a probability p_i to be released, for each $i \in \{1, .., k\}$. We can thus have, at a given time instant, between 0 and k jobs that are released simultaneously.
- c_i is the size of J_i, randomly chosen in the set $\{0, .., 4\}$.
- d_i is the deadline of J_i, randomly chosen in the set $\{1, 2\}$.
- $k \leq 5$ and $\sum_{i=1}^{k} p_i \in [0.8, 1]$. These two choices ensure that the generated systems are "interesting" for the learning phase, *i.e.*, they have an average load such that the range of feasible speeds in not reduced to a singleton (as it would be the case, for instance, if at each instant the cumulated size of job arrivals would amount to a load equal to $s_{\max}$).

Moreover, the chosen value of the discount γ is 0.9.

Fig. 7.2 displays the evolution of the $\mathrm{span}(\hat{V}^{(\mathrm{SQL})} - V^*)$, where $\hat{V}^{(\mathrm{SQL})}$ is the energy cost learned after n iterations, n being the length of the training period, ranging between 10^4 and 10^7. We use the span to check convergence instead of some norm on $\hat{V}^{(\mathrm{SQL})} - V^*$, because when the span is 0, policy (SQL) is exactly an optimal policy.

This result shows us that we are far from the theoretical convergence even after 10^7 learning steps, but *in practice* the overhead results are close from the optimal value.

Fig. 7.3 depicts the overhead of the energy consumption of (OA) compared with the speed policy obtained by (SQL), in function of the duration n of the learning phase. As can be seen, the convergence evolves with learning: the speed policy learned with (SQL) outperforms (OA) by 6.4% on average after 10^4 learning steps and this percentage progresses when n increases. Indeed this percentage is of 7.6% on average after 10^7 learning steps.

Between 10^4 and 10^7 learning steps, all the box-plots have almost the same distribution: their median value is 5.6%, and their quartiles are also identical. Only their mean values differ, due to the extremal values, and so due to some particular job sets, with a negative overhead when we compare to (OA). Finally, once $n \geq 5.10^6$, all the overheads are strictly positive, meaning that for each set of jobs, the speed policy learned with (SQL) outperforms (OA). Taken 5.10^6 iterations is a worse result than this one observed in Chapter 6. This observation will be developed in Section 7.7.6.

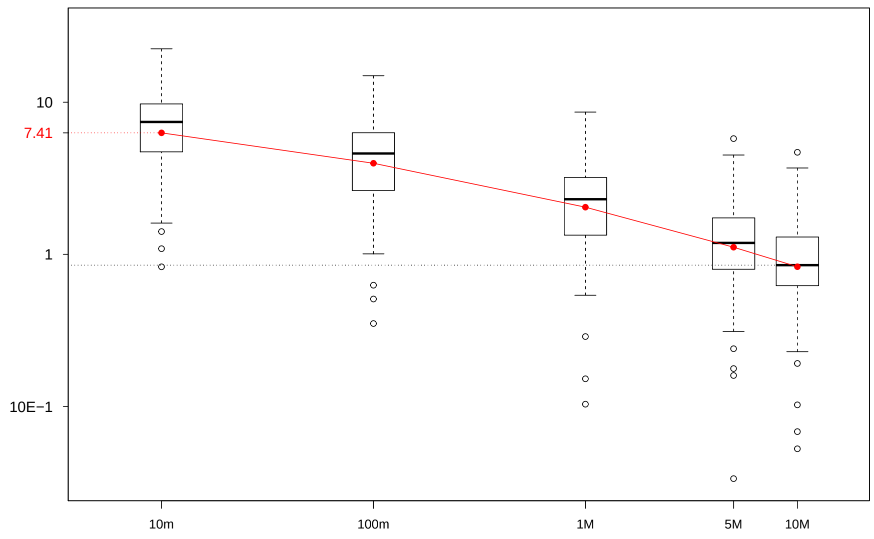


Figure 7.2.: Evolution of $\mathrm{span}(\hat{V}^{(\mathrm{SQL})} - V^*)$ in function of the duration n of the learning phase: $n \in \{10^4, 10^5, 10^6, 5.10^6, 10^7\}$ for 70 job streams. Even after 10^7 learning steps, the $\mathrm{span}(\hat{V}^{(\mathrm{SQL})} - V^*)$ value is still significant.

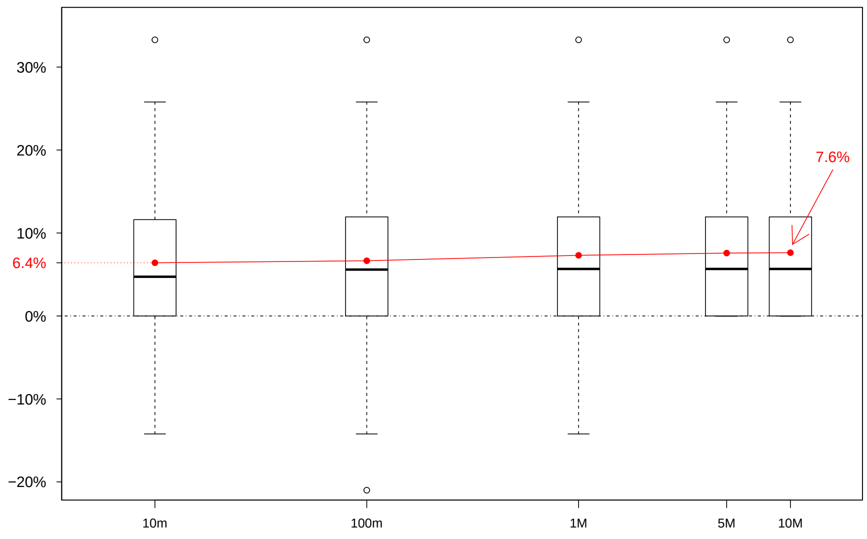


Figure 7.3.: Overhead in percentage of (OA) versus the Q-learning algorithm (SQL) depending on the duration n of the learning phase: $n \in \{10^4, 10^5, 10^6, 5.10^6, 10^7\}$.

Like in Chapter 6, it is interesting to compare Fig. 7.2 and Fig. 7.3: even though the convergence is not reached after 10^7 learning steps for (SQL), the energy consumption of the speed policy learned with (SQL) is always strictly better than that of (OA) after 5.10^6 learning steps.

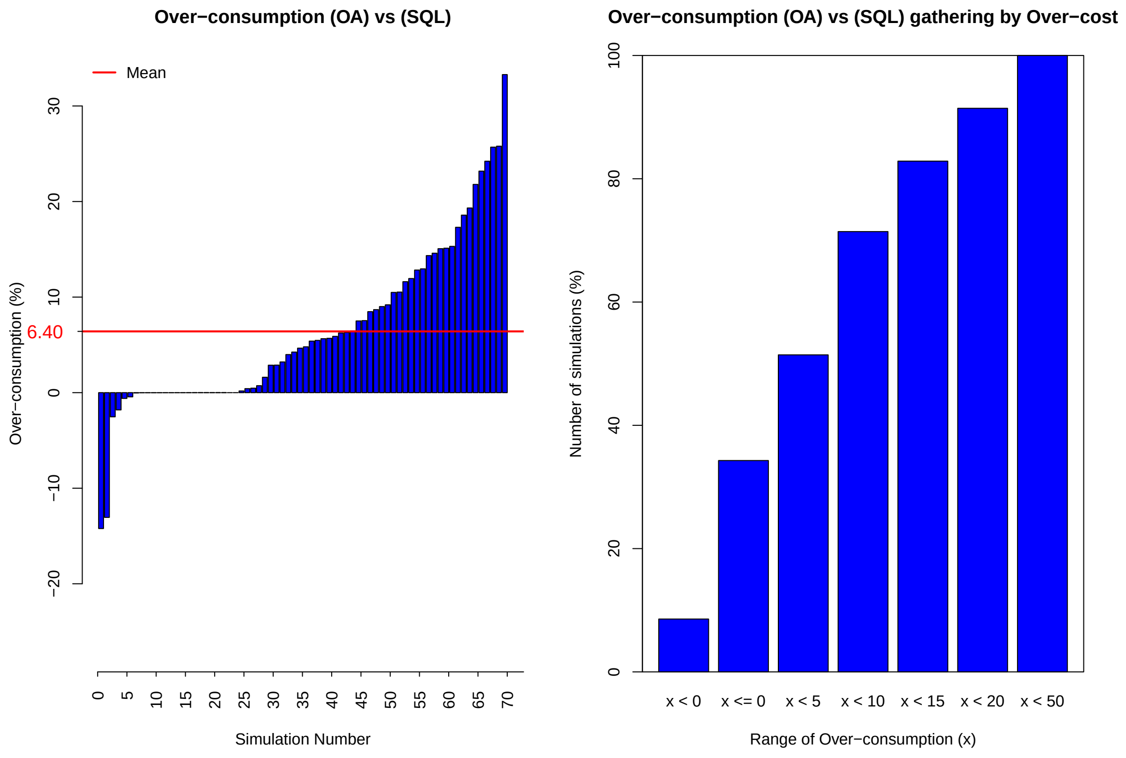


Figure 7.4.: Energy consumption overhead in percentage of (OA) versus the speed policy learned with (SQL) after 10^3 learning steps. The job characteristics (p_i, c_i, d_i) are such that $p_i \in [0.1, 0.9]$, $c_i \in \{0, \dots, 4\}$, and $d_i \in \{1, 2\}$.

Fig. 7.4 depicts the energy consumption overhead in percentage of (OA) versus the speed policy learned with (SQL) after 10^3 learning steps. The job characteristics $J_i = (p_i, c_i, d_i)$ are chosen as follows: the arrival probabilities p_i are randomly chosen in the interval $[0.1, 0.9]$, the sizes c_i are randomly chosen in the set $\{0, \dots, 4\}$; and the deadlines d_i are randomly chosen in the set $\{1, 2\}$.

Fig. 7.4-left shows the energy overhead of each individual job set sorted by the resulting energy consumption overhead. (OA) outperforms (SQL) over six job sets. Otherwise (SQL) systematically outperforms (OA), and the average gain is 6.4%. (SQL) is equivalent to (OA) for 35% job sets.

Fig. 7.4-right gathers the simulations by their over-consumption percentage value. Each vertical bar corresponds to an interval of energy consumption overhead of (OA) versus (SQL). For instance, there are 53% job sets for which this energy overhead is below 5%.

Experiment 3: Random family of job streams with Δ = 3

Now in the case of (SQL), we study the overhead and the number of iterations before convergence of several random systems, with a different maximal deadline.

The first set of simulation are done for $\Delta = 3$ and 8 random systems. The cost of the policy obtained by (SQL) is always better than the cost obtained by (OA), and is always equal to the cost of the optimal policy (VI). Indeed, the mean of the overhead of (OA) versus Q-learning is 8.19%.

However one problem remains, as in the previous case, after 10^8 iterations, the convergence between $\hat{q}(w) = \min_s \hat{Q}(w,s)$ and the value of $V^*_{(\mathrm{VI})}$ is not reached. The percentage of difference between the two values is above 5% for 7 systems out of 8.

The second set of simulations are done for $\Delta = 3$ and 20 random systems. As in the previous case, the cost of the policy obtained by Q-learning is always better than this one obtained by (OA), and is always equal to the cost of the optimal policy (VI). The mean value is 6.08%. In terms of convergence of $\hat{q}(w)$, the result is better, in the sense that 14 systems out of 20 have converged at 5%.

7.7.5 Regret Analysis (AQL)

In the (AQL) algorithm, we compute the regret, as defined in Section 7.7.1. This analysis is done for the job stream described in Ex. 7.1. A good curve regret has to show that the regret increase should be sub-linear, due to the improvement of the choices made by the learning algorithm. In our case, we note that all curves are linear in time and it shows that even if the time grows, the policy does not really improved at least over the horizon of the simulation. This seems to indicate that Q-learning is not efficient to minimize the regret in our framework. It could be interesting to investigate other techniques as UCB method based on Q-value (see [PT17]).

7.7.6 Comparison with the Transition Probability Matrix

We notice that in the previous Chapter, Chapter 6, on the same set of experimentation, we converge quickly to the optimal solution than in the Q-learning solution we present in this chapter. Indeed after one thousands samples $n = 1000$, the P_n matrix learned lead to a best overhead for (OA) versus (PL) than the overhead of the speed policy learned with (SQL). The Q-learning algorithm reached the same overhead in comparison with (PL) only after 10^7.

7.8 Conclusion

Model-free reinforcement learning algorithms, such as Q-learning, allow us to compute efficient speed policies to process real-time jobs while minimizing the energy consumption. The technique

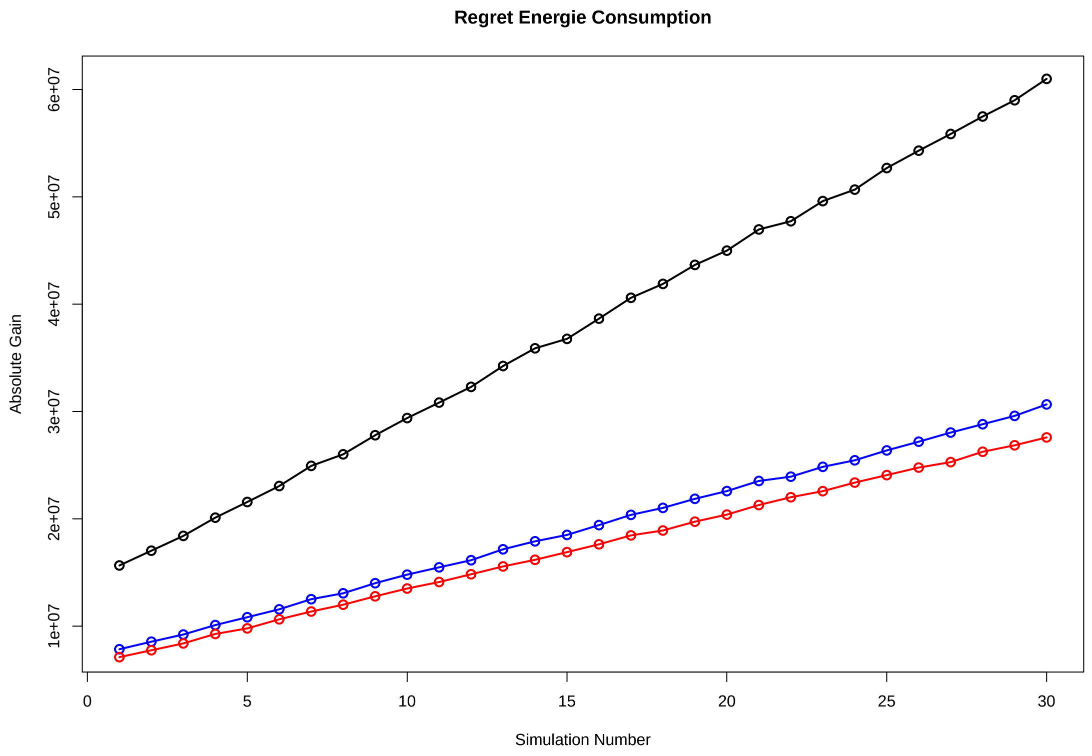


Figure 7.5.: Regret for (OA), Q-learning with 2 speeds, Q-learning with 3 speeds in comparison with (MDP)

studied here, Q-learning, takes significant time to converge (*i.e.*, even after 10^7 learning steps, the convergence of the Q value is not reached yet), but the optimal policy is actually reached much faster (*i.e.*, after 5.10^6 learning steps the learned speed policy outperforms (OA) on all randomly chosen job sets).

The results obtained in Chapter 6 by learning the probability transition matrix are better than those obtained in this chapter. This means that it is more efficient to learn the probability transition matrix (and then to compute the optimal speed policy) than to learn directly the optimal speed policy.

In terms of future research directions, it will be interesting to investigate state space reduction techniques. Another possibility will be to generalize Chapter 5 by studying learning method for non clairvoyant active jobs.

The present chapter ends our journey started in Chapter 3 with a total knowledge of the real-time system and ending in Chapter 7 with no knowledge at all. It is now time to study the exact condition on the maximal speed of the processor under which our online speed policies are feasible, and to compare these feasibility conditions with those of the other online speed policies from the literature. This will be the topic of the next chapter.

8 Feasibility of online speed policies

This chapter is different from the others, in the sense that it does not focus on the energy problem, but it explores the notion of feasibility, that has been quickly studied in previous chapters. Furthermore, unlike the rest of the document, techniques used in this chapter are totally different from what was done before. The feasibility of our speed policy and also different policies that exist in the literature are studied in this chapter. The feasibility is the ability for the processor to execute any sequence of jobs while satisfying the two following constraints:

1. The processor speed is always below the maximal processor speed $s_{\max}$.
2. No job misses its deadline.

One policy is the policy we present in Chapter 4. We will begin by presenting in Section 8.1 the different policies we will analyse, and then in Section 8.2 the previous work done on policy feasibility.

Before presenting the state of the art section, Section 8.1, I would like to emphasize strongly on the point that in this extension the notation is slightly different, in comparisons to the rest of this thesis, to make all the equations more readable: the time considered in the remaining work function $w_t(.)$, defined in Chapter 4 is absolute. This modification is only valid for this chapter.

This chapter is published in Real Time System Journal (RTSJ) [GGP20b]

8.1 State of the Art

Let us recall how we consider a job in this chapter, and what is the problem on which we focus on: each job is characterized by its arrival time, its size *i.e.* the amount of work to complete the job, and its strict deadline, either defined absolutely or relatively to the arrival time. We consider the particular case of unconstrained HRTS executed on a single core processor with variable processor speed. An HRTS is therefore characterized by a tuple $(C, \Delta, s_{\max})$, where C is the maximal size of the jobs, Δ is their maximal deadline, and $s_{\max}$ is the maximal speed of the processor. The inter-arrival times between the jobs are unconstrained (*i.e.* neither periodic or sporadic).

Changing the speed of the processor can help to reduce the energy consumption of the processor, which is essential in many embedded systems. In fact, this is the reason why modern processors are equipped with Dynamic Voltage and Frequency Scaling (DVFS) capabilities. Several speed selection policies have been proposed to save energy by modifying the speed of the processor online. The main idea behind all online speed policies is to lower the speed when the current load is low, in order to save energy and, when the load is high, to increase the speed to execute all jobs before their deadlines.

In this chapter, the main goal is to analyze the feasibility of existing online speed policies. A policy is feasible if and only if each job is executed before its absolute deadline. Without loss of generality, we assume that the time scale is discrete and that a new job arrives at each time step. In contrast, the processor speed can change at any time.

The first online speed policy that comes to mind involves, at each time step, executing entirely the active job within one time step. Obviously this policy is feasible, because all the jobs finish before their deadline. Moreover, the maximal processor speed used under this policy is not larger than C. Therefore, this policy is feasible if $s_{\max} \geq C$. This is optimal in terms of feasibility because no policy can be feasible when $s_{\max} < C$: indeed, if $s_{\max} < C$, then a job of size C with deadline 1 will miss its deadline. In contrast, regarding the energy consumption, this policy consumes more than any other policy because it does not take advantage of job deadlines (assuming that the energy is an increasing convex function, which is usually the case). For these reasons, we analyze in this chapter the feasibility of known policies that lower the energy consumption.

We investigate the four following online speed policies. To the best of our knowledge, these are the four such existing speed policies. The first two ones are (AVR) and (OA), both from [YDS95], which both try to optimize the energy consumption of a real-time system. The third one is (BKP) from Bansal et al. [BKP07], the goal of which is to improve the competitive ratio of (OA). The fourth one is a Markov Decision Process policy called (MP) in the rest of the chapter, which optimizes the expected energy consumption when statistical information on the arrival, WCET, and deadline of the jobs are available [GGP17].

In their original respective chapter, the authors of (AVR), (OA), and (BKP) all make the unrealistic assumption that $s_{\max}$ is unbounded, *i.e.* $s_{\max} = +\infty$. Under this assumption, feasibility is not as problematic: all jobs can be executed before their deadline as long as the current selected speed is large enough. However, under the more realistic assumption of a bounded $s_{\max}$, one needs to compute the feasibility region in the parameter space of $(C, \Delta, s_{\max})$. Our goal in this chapter is therefore to determine, for the classical policies (AVR), (OA), (BKP), and for (MP), the maximal speed $s_{\max}$ as a function of C and Δ, that ensures feasibility, and to compare the four policies in this respect.

The chapter is organized as follows. We survey the related work in Section 8.2. Then we present the job model used in Section 8.3 and formulate the feasibility analysis problem in Section 8.4. In the subsequent sections we analyze each online speed policy and we prove, for each of them, what is the smallest value of $s_{\max}$ that ensures feasibility (Sections 8.5 to 8.8). Finally, we compare the four online speed policies based on these values $s_{\max}$ in Section 8.9 before concluding in Section 8.10.

8.2 Related work

The work that is most closely related to our is [CST09], which investigates the feasibility of (AVR) and (OA) (this latter speed policy being called (OPT) in their paper). The system model is a single-core processor that must execute an infinite sequence real-time jobs, specified by an *arrival curve* (as in the Real-Time Calculus [TCN00]). Arrival curves generalize both the periodic task model and the sporadic task model with minimal inter-arrival time. An important assumption in [CST09] is that all the jobs have the same WCET C and the same relative deadline Δ. The main result is that, both for (OA) and (AVR), the feasibility condition is $s_{\max} \geq \frac{\alpha^u(\Delta)}{\Delta}$, where α^u is the upper arrival curve of the sequence of jobs, meaning that $\alpha^u(D)$ is an upper bound on the work that can arrive during any time interval of length D. Actually, the same feasibility condition applies to (BKP) and (MP), although these speed policies are not studied in [CST09]. In contrast to this result, we do not constrain the jobs to have the same WCET nor the same deadline. Therefore the analysis becomes completely different as well as the feasibility conditions which are now different for each policy.

To the best of our knowledge, all the other results on feasibility analysis of online speed policies found in the literature target system models either with a fixed inter-arrival time between the jobs (*i.e.* periodic tasks) or with a bounded inter-arrival time (*i.e.* sporadic tasks). Papers in this category are plentiful, let us just cite [JG04] in the periodic case and [AIS04] in the sporadic case. In contrast, we make no assumption on the inter-arrival times between jobs.

8.3 Presentation of the problem

8.3.1 Hard real-time systems

Let us recall the system we consider. As previously we consider a HRTS that executes an infinite sequence of sporadic and independent jobs $\{J_i\}_{i\in\mathbb{N}}$ on a single-core processor with varying frequency. Each job J_i is defined as a tuple (r_i, c_i, D_i) where $r_i \in \mathbb{N}$ is the release time (or arrival time), $c_i \in \mathbb{N}$ is the size (also called workload), *i.e.* the amount of work to complete the job, and $D_i \in \mathbb{N}$ is the absolute deadline of job J_i, satisfying $D_i > r_i$. The jobs are ordered by their release times. Their relative deadlines are $d_i := D_i - r_i$, *i.e.* the amount of time given to the processor to execute the job. The jobs are sporadic, meaning that their arrival times do not follow any particular pattern. This is the most general model of jobs.

We further assume that all jobs have a bounded relative deadline: there exists Δ such that

$$\forall i, d_i = D_i - r_i \leq \Delta \tag{8.1}$$

where Δ is the maximal relative deadline. Several jobs may arrive simultaneously but in any case the cumulated size is assumed to be bounded by C. In other words:

$$\forall t, \sum_{i \mid r_i = t} c_i \leq C. \tag{8.2}$$

Finally, we denote by $\mathcal{J}_{C,\Delta}$ the set of all possible sequences of jobs that satisfy the two assumptions stated in Eqs. (8.1) and (8.2).

Definition 8.1 (Set of all possible sequences of jobs: $\mathcal{J}_{C,\Delta}$).

$$\mathcal{J}_{C,\Delta} := \left\{ J = \{J_i = (r_i, c_i, D_i)\}_{i\in\mathbb{N}} \middle| \forall t, \sum_{i\,|\,r_i=t} c_i \leq C \;\wedge\; \forall i, D_i - r_i \leq \Delta \right\}. \tag{8.3}$$

Minimality of the assumptions. Let us anticipate a bit on what follows and comment about the relevance of the two assumptions stated by Eq. (8.1) and (8.2). We claim that these are the minimal assumptions under which feasibility of a speed policy can be asserted.

First, in most practical cases, the set of jobs comes from a finite set of tasks (infinite sequences of jobs with the same features). In this case, relative deadlines and sizes are always bounded. Besides, if the set of jobs is finite, then everything is bounded.

Consider now the most general case, *i.e.* with an infinite set of sporadic jobs. If the relative deadlines are not bounded, then the set of pending jobs at some arbitrary time t cannot be bounded and the time needed to compute the current speed for all online policies is also unbounded, so that feasibility cannot be asserted in finite time.

Once the condition that all jobs have a bounded deadline is stated, the assumption on the arriving work (8.2) must also be made. Indeed, if a set of jobs arrives at time t, all with deadlines bounded by Δ, and brings an unbounded amount of work into the system, then no speed policy with a given maximal speed will be able to execute this work before time $t + \Delta$.

8.3.2 Scheduling policy

At any time $t \in \mathbb{R}$, several jobs may be active (*i.e.* released and not yet finished). In this case we must choose which job to execute first on the single-core processor. This ordering is known as a *schedule* and the policy for making this choice is known as the *scheduling policy*. Let us recall the definition of a schedule feasibility.

Definition 8.2 (Schedule feasibility). *A schedule is* feasible *over an infinite sequence of jobs* $J = \{(r_i, c_i, D_i)\}_{i\in\mathbb{N}} \in \mathcal{J}_{C,\Delta}$ *if and only if each job* (r_i, c_i, D_i) *is executed between its release time and its absolute deadline,* i.e. *between* r_i *and* D_i.

It has been shown that the Earliest Deadline First (EDF) scheduling policy is optimal for feasibility [LL73], meaning that if a sequence J is feasible for some scheduling policy, then it is also feasible under EDF. Therefore, in the following, we will always assume that the processor uses EDF to schedule its active jobs.

8.3.3 Online speed policy

Let us recall the goal we reach in this chapter: we focus on online speed policies, the goal of which is to choose, at each time t, the speed at which the processor should run, based on the current information (we assume that no look-ahead is available).

Let us given a sequence of jobs $J = \{(r_i, c_i, D_i)\}_{i\in\mathbb{N}}$. We note also in this chapter the speeds $s(t)$, that corresponds to the speed used at all time $t \in \mathbb{R}$.

As in all the document, all the release times, job sizes, and deadlines are integer numbers. Therefore, the sequence of jobs $\{J_i, r_i \leq t\}$ only changes at integer time instants. This is not the case for the processor speeds $\{s(u), u \leq t\}$, which can change at any time instant. We will detail this in Section 8.3.4.

Let us recall the definition of an online speed policy:

Definition 8.3 (Online speed policy). *An online* speed policy π *is a function that assigns, at time* t *with the history* $\mathcal{H}_t$*, a speed* s *to the processor:*

$$\pi(\mathcal{H}_t, t) = s. \tag{8.4}$$

where $\mathcal{H}_t$ is the history, notion defined in Chapter 2.

In the following, we will often use $\pi(t)$ to simplify the notation, but one should keep in mind the fact that, in full generality the speed selected at time t may depend on t, the jobs that arrived before t, and the speeds selected before t.

Since the maximal speed of the processor is $s_{\max}$, any speed policy π must satisfy the following constraint:

$$\forall t, \forall J,\ 0 \leq \pi(\mathcal{H}_t, t) \leq s_{\max}. \tag{8.5}$$

8.3.4 Speed decision instants

We recall here the definition of the *speed decision instants*: It is the instants at which the processor speed can change. These times do not necessarily coincide with the job arrival times. For instance, processor speeds may change several times between two potential job arrivals. In the rest of this chapter, we study the two different cases:

- The processor speed changes can only occur when a job arrives: $t \in \mathbb{N}$.
- The processor speed changes can occur at any time: $t \in \mathbb{R}$.

In the following, we denote by $\mathcal{T}$ the set of speed decision instants. As discussed above, the two possible cases are studied in this chapter: $\mathcal{T} = \mathbb{N}$ and $\mathcal{T} = \mathbb{R}$. For (OA), (AVR), and (MP), we will show that the cases $\mathcal{T} = \mathbb{N}$ and $\mathcal{T} = \mathbb{R}$ yield the same feasibility conditions. For (BKP), the two cases are slightly different.

8.3.5 Feasibility problem for online speed policies

The goal of this chapter is to determine the condition for which feasibility is satisfied for several speed policies. To do that, we will recall the definition of the speed policy's feasibility.

Definition 8.4 (Speed policy's feasibility)**.** *An* online speed policy π *is feasible over an infinite sequence of jobs* $J = \{(r_i, c_i, D_i)\}_{i \in \mathbb{N}}$ *if and only if when the processor runs at speed* $\pi(t)$ *for all* t *and uses EDF, each job* (r_i, c_i, D_i) *is executed before its absolute deadline:*

$$\pi \textit{ is feasible} \iff \left(\sup_{J \in \mathcal{J}_{C,\Delta}} \sup_{t \in \mathcal{T}} \pi(t) \leq s_{\max} \right) \wedge \textit{ no missed deadline}. \tag{8.6}$$

In Eq. (8.6), the second term "no missed deadline" is not very explicit. For this reason we redefine it by using the remaining work function, which is presented next. In the rest of the chapter we use the following notation: x_+ is the positive part of x: $x_+ := \max(x, 0)$.

It should be noted that the definition below is a more general definition for the remaining work function, because it is considered that speeds can change at any time to place this chapter in the broadest situation as analysed in Appendix A of Chapter 4.

Definition 8.5 (Remaining work function)**.** *The* remaining work function *under* π *at time* t *is the function* $w_t^\pi(.)$, *such that, at any future time* $u \geq t$, *the* remaining work $w_t^\pi(u)$ *is the amount of work that has arrived by time* t *whose deadline is before* u, *minus the amount of work already executed at time* t. *It satisfies a Lindley's equation by induction:*

$$\begin{cases} w_0^\pi(u) = 0 & \forall u \geq 0 \\ w_t^\pi(u) = \left(w_k^\pi(u) - \int_k^t \pi(v)dv \right)_+ + A(t,u) & \forall k \in \mathbb{N} \textit{ with } k < t \leq k+1 \\ & \textit{and } \forall u \geq t > 0 \end{cases} \tag{8.7}$$

where $A(t,u)$ *is the amount of work corresponding to the jobs arriving at time* t *whose deadline is smaller or equal to* u.

Two remarks are in order:

Remark 8.1. *The arrival function* $A(t,u)$ *is equal to* 0 *if* $t \notin \mathbb{N}$, *because the release times of all jobs are in* $\mathbb{N}$.

Remark 8.2. *Since the maximal job relative deadline is* Δ, $w_t^\pi(t+\Delta)$ *is the total amount of remaining work at time* t. *In other words,* w_t^π *increases up to time* t_Δ *and stays constant after that time* $t + \Delta$: $\forall u \geq t+\Delta$, $w_t^\pi(u) = w_t^\pi(\Delta+t)$. *Moreover, for any online policy* π, $\int_k^{k+1} \pi(v)dv \leq w_k^\pi(k+\Delta)$ *because, at time* k, *the processor can only execute work present in the system at time* k. *By straightforward induction, this implies:*

$$w_t^\pi(t+\Delta) = \sum_{r_i \leq t} c_i - \int_0^t \pi(v)dv \tag{8.8}$$

and when no deadlines are missed, then:

$$w_t^\pi(t+\Delta) \leq C\Delta. \tag{8.9}$$

Feasibility Characterization

Using Def. (8.7) of the remaining work function, one can make the definition of feasibility given in Def. (8.4) more explicit. For this purpose, we state Prop. 8.1 that links the remaining work function and the policy. This proposition introduces a new condition of feasibility.

Proposition 8.1.

$$\pi \textit{ is feasible} \iff \left(\sup_{J \in \mathcal{J}_{C,\Delta}} \sup_{t \in \mathcal{T}} \pi(t) \leq s_{\max} \right) \wedge \left(\forall J \in \mathcal{J}_{C,\Delta}, \forall t \in \mathcal{T},\ w_t^{\pi}(t) = 0 \right). \tag{8.10}$$

The first condition says that the speed selected by π at t must always be smaller than $s_{\max}$, while the second condition says that at any t, all the work whose deadline is before t has already been executed. Although this may seem trivial, let us write an explicit proof of this equivalence.

Proof. We rely on the definition of feasibility given in Def. 8.4. There are two parts in this definition, and to prove the proposition, we will begin to show that:

$$\text{no missed deadline} \iff \forall J \in \mathcal{J}_{C,\Delta}, \forall t \in \mathcal{T},\ w_t^{\pi}(t) = 0. \tag{8.11}$$

The proof of Eq. (8.11) is divided in two parts, each of them proves one implication.

1. No missed deadline $\implies \forall t, 0 = w_t^{\pi}(t)$:

   By contraposition, let us show that $0 < w_t^{\pi}(t) \implies$ missed deadline. If $0 < w_t^{\pi}(t)$, then it means that some work whose deadline is before t has not been executed by time t, so at least one job has missed its deadline before time t.

2. $\forall t, 0 = w_t^{\pi}(t) \implies$ no missed deadline:

   If $0 = w_t^{\pi}(t)$, then at each time t, all the work whose deadline was before t has been executed. Thanks to EDF, we know that all the jobs whose deadline is exactly at time t have been executed before t. This is true for all t, so it is also true for all the jobs.

The condition involving $s_{\max}$ is the same as in the original definition. □

The following proposition establishes a necessary condition of the feasibility for any online speed policy π.

Proposition 8.2. *For decision instants $\mathcal{T} = \mathbb{N}$ and $\mathcal{T} = \mathbb{R}$ and for any policy π, a necessary condition of feasibility is:*

$$s_{\max} \geq C. \tag{8.12}$$

Proof. Let π be any feasible online speed policy and let J be the sequence of jobs made of the single job $J_0 = (0, C, 1)$. By Def. 8.5, $w_1^{\pi}(1) = (C - \int_0^1 \pi(v)dv)_+$. The second part of the feasibility condition of π says that at time 1, $w_1^{\pi}(1)$ must be equal to 0. This implies $\int_0^1 \pi(v)dv \geq C$. Since $\int_0^1 \pi(v)dv \leq \max_{0 \leq t \leq 1} \pi(t)$, we therefore have, $\max_{0 \leq t \leq 1} \pi(t) \geq C$. Then, the first part of the feasibility condition implies that $s_{\max} \geq \max_{0 \leq t \leq 1} \pi(t)$. Putting both parts together yields $s_{\max} \geq C$. □

Proposition 8.3. *In the case of integer decision instants ($\mathcal{T} = \mathbb{N}$), the condition $\forall t \in \mathbb{R},\ w_t^\pi(t) = 0$ can be re-written as $\forall k \in \mathbb{N},\ \pi(k) \geq w_k^\pi(k+1)$.*

Proof. The proof simply follows the definitions. When the speed is constant in the interval $[k, k+1)$,

$$w_{k+1}^\pi(k+1) \quad = \quad (w_k^\pi(k+1) - \pi(k))_+ + A(k+1, k+1),$$

with $A(k+1, k+1) = 0$ because jobs arriving at time $k+1$ have a deadline at least $k+2$. Hence:

$$w_{k+1}^\pi(k+1) \quad = \quad (w_k^\pi(k+1) - \pi(k))_+$$

It follows that $w_{k+1}^\pi(k+1) = 0$ if and only if $(w_k^\pi(k+1) - \pi(k))_+ = 0$. By definition of the $\max$, this is equivalent to $\pi(k) \geq w_k^\pi(k+1)$. □

8.4 Feasibility analysis

The goal of this chapter is to study the feasibility of the four different online speed policies (OA), (AVR), (BKP), and (MP). For each policy, we formally establish a necessary and sufficient feasibility condition on $s_{\max}$. In each case, the proof follows the same route. We first check that if $s_{\max} = \infty$ then the policiy is feasible. This part of the proof is already provided in the papers introducing the policies, but we briefly sketch them when the argument is trivial. Then, still assuming that $s_{\max} = \infty$, we compute the maximal speed $\overline{\pi}$ used by the online policy under a worst case sequence of jobs in $\mathcal{J}_{C,\Delta}$. Therefore, a necessary and sufficient condition of feasibility is $s_{\max} \geq \overline{\pi}$. We construct such a worst case sequence for each policy. While these worst case sequences will look similar (at least the first three), the analysis relies on very different techniques:

- The proof for (OA) policy uses a construction (Lindley's equation, with a backward construction) that comes from queueing theory (Section 8.5).
- The proof for (AVR) is based on the explicit construction of a worst case, which consists of a maximal number of jobs that have the same deadline (Section 8.6).
- The proof for (BKP) exploits arithmetic considerations (Section 8.7).
- The proof for (MP) is based on a dynamic programming analysis (Section 8.8).

At any time t, the (OA) and (MP) policies both compute the processor speed based on the work remaining at t, while the (AVR) and (BKP) policies do not. This is in part why the proofs are so diverse. As a final note before starting with the proofs, the case of (OA) is by far the more interesting. In spite of the apparent simplicity of (OA), the proof uses several backward inductions as well as properties of generalized differential equations (with non-differentiable functions).

8.5 Feasibility of the Optimal Available speed policy (OA)

8.5.1 Definition of (OA) [YDS95]

Definition 8.6 (Optimal Available (OA))**.** *At each time $t \in \mathcal{T}$, the job that has the earliest deadline is executed at speed:*

$$\pi^{(\mathrm{OA})}(t) = \max_{v>t}\left(\frac{w_t^{(\mathrm{OA})}(v)}{v-t}\right) \tag{8.13}$$

where $w_t^{(\mathrm{OA})}(.)$ is the remaining work defined in Def. 8.5.

To illustrate (OA), let us consider the following set of jobs with $\mathcal{T} \in \mathbb{N}$, which is composed of 3 jobs and belongs to $\mathcal{J}_{4,5}$:

- $J_1 = (r_1 = 0, c_1 = 1, d_1 = 4)$ hence $D_1 = 4$,
- $J_2 = (r_2 = 3, c_2 = 4, d_2 = 6)$ hence $D_2 = 3$,
- $J_3 = (r_3 = 3, c_3 = 1, d_3 = 8)$ hence $D_3 = 5$,

Let us compute the (OA) speed at time 3. According to Eq. (8.13), it is equal to:

$$\pi^{(\mathrm{OA})}(3) = \max_{v>3}\left(\frac{w_3^{(\mathrm{OA})}(v)}{v-3}\right)$$

At each of the three instants 0, 1, and 2, only the job J_1 is present, so the speed computed by Eq. (8.13) is equal to:

$$\pi^{(\mathrm{OA})}(0) = \pi^{(\mathrm{OA})}(1) = \pi^{(\mathrm{OA})}(2) = \frac{c_1}{D_1} = \frac{1}{4}.$$

Therefore, at time 3, we have:

$$\begin{aligned}
\pi^{(\mathrm{OA})}(3) &= \max \left\{\frac{w_3^{(\mathrm{OA})}(4)}{d_1-3}, \frac{w_3^{(\mathrm{OA})}(6)}{d_2-3}, \frac{w_3^{(\mathrm{OA})}(8)}{d_3-3}\right\} \\
&= \max \left\{\frac{c_1 - \pi^{(\mathrm{OA})}(0) - \pi^{(\mathrm{OA})}(1) - \pi^{(\mathrm{OA})}(2)}{d_1-3},\right. \\
&\qquad \frac{c_1 + c_2 - \pi^{(\mathrm{OA})}(0) - \pi^{(\mathrm{OA})}(1) - \pi^{(\mathrm{OA})}(2)}{d_2-3}, \\
&\qquad \left.\frac{c_1 + c_2 + c_3 - \pi^{(\mathrm{OA})}(0) - \pi^{(\mathrm{OA})}(1) - \pi^{(\mathrm{OA})}(2)}{d_3-3}\right\} \\
&= \max \left\{\frac{1}{4}, \frac{17}{12}, \frac{21}{20}\right\}
\end{aligned}$$

In conclusion, we have $\pi^{(\mathrm{OA})}(3) = \frac{17}{12}$.

8.5.2 Feasibility analysis of (OA)

In this section, we will determine the smallest maximal processor speed $s_{\max}$ that guarantees the feasibility of (OA). Theorem 8.1 gives a necessary and sufficient feasibility condition for (OA).

Theorem 8.1. (OA) *is feasible* $\iff s_{\max} \geq C(h_{\Delta-1}+1)$, *where* h_n *is the* n*-th harmonic number:* $h_n = \sum_{i=1}^{n} 1/i$.

Proof. We distinguish the cases where the speed decision instants are integer and real numbers.

♦ **The speed decision instants are integer numbers:** $\mathcal{T} = \mathbb{N}$.
In the integer case, Eq. (8.13) becomes:

$$\pi^{(\mathrm{OA})}(t) = \max_{v\in\mathbb{N},\, v>t} \left(\frac{w_t^{(\mathrm{OA})}(v)}{v-t} \right). \tag{8.14}$$

By taking $v = t+1$, Eq. (8.14) implies that $\pi^{(\mathrm{OA})}(t) \geq w_t^{(\mathrm{OA})}(t+1)$. Therefore, the feasibility Equation (8.10) can be written as a condition on $s_{\max}$ only:

$$(\mathrm{OA}) \text{ is feasible} \quad \iff \quad \forall t,\ s_{\max} \geq \pi^{(\mathrm{OA})}(t).$$

The rest of the proof is structured as follows. (i) We will first derive a bound on $\pi^{(\mathrm{OA})}(t)$ (steps 1, 2, and 3). (ii) Then we will construct an explicit worst-case scenario that reaches this bound asymptotically.

Let us first compute an upper bound on the remaining work $w_t^{(\mathrm{OA})}(v)$, for any $t \in \mathbb{N}$ and any integer $v > t$. This will be done in several steps. To simplify notations, in the following, we denote $\pi^{(\mathrm{OA})} = \pi$ and $w^{(\mathrm{OA})} = w$, since the only speed policy considered hare is (OA) and no confusion is possible.

We can focus on times $v \leq t+\Delta$ because the remaining work after time $t+\Delta$ remains the same (see Remark 8.2). Now, $w_t(v)$ only depends on three things:

- the remaining work function at time $v-\Delta$: $w_{v-\Delta}(.)$,
- the work that arrives between times $v-\Delta+1$ and t,
- and the speeds used at times $v-\Delta$ to $t-1$.

The definition of $w_t(v)$ yields:

$$\begin{aligned} w_t(v) &= \big(w_{t-1}(v) - \pi(t-1)\big)_+ + A(t,v) && (8.15)\\ w_{t-1}(v) &= \big(w_{t-2}(v) - \pi(t-2)\big)_+ + A(t-1,v) && (8.16)\\ &\ \vdots \\ w_{v-\Delta+1}(v) &= \big(w_{v-\Delta}(v) - \pi(v-\Delta)\big)_+ + A(v-\Delta+1,v). && (8.17) \end{aligned}$$

This first shows that the function w_t increases when π decreases.

Step 1: The first step amounts to showing that $w_t(v)$ becomes larger if the sizes of all the jobs whose absolute deadline is larger than v are set to 0, while keeping the rest unchanged.

This fact is easy to check: In Eqs (8.15)-(8.17), the only terms that depend on those jobs are the speeds. Under (OA), the speeds are increasing with the remaining work. Therefore, by removing these jobs, all the speeds are decreased (or remain the same) and $w_t(v)$ is increased.

Step 2: The second step amounts to checking that, if the remaining work function $w_{v-\Delta}(.)$ is replaced by the function $w^*_{v-\Delta}(.)$ such that (i) $w^*_{v-\Delta}(i) = 0$ for $i = v - \Delta + 1, \ldots, v - 1$ and $w^*_{v-\Delta}(v) = w_{v-\Delta}(v)$, and such that (ii) all jobs arriving at times $v - \Delta < i \leq t$ have their deadline set at v, then this change increases the remaining work at time v. This construction is illustrated in Fig. 8.1 where the $w^*_{v-\Delta}(.)$ function is depicted by the black curve.

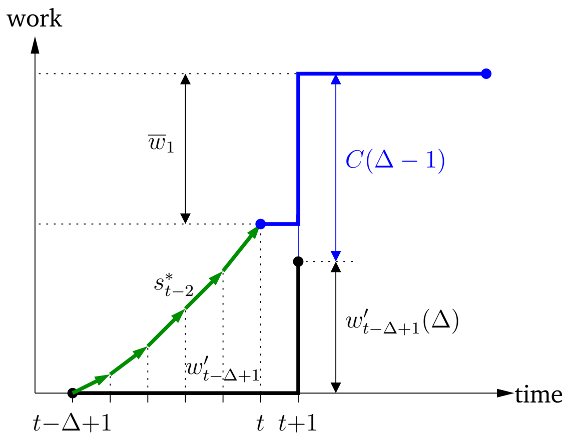


Figure 8.1.: Construction of $w^*_t(v)$ for $v = t + 1$ and $\Delta = 6$. The bold black curve is the lower bound on the remaining work $w^*_{v-\Delta}(.)$. The bold blue curve is the final upper bound $\overline{w}_1$ on the remaining work. The bold green arrows represent the work executed by the processor at each time slot i at speed $\pi^*(i)$.

We will show this by induction (putting a star on all values computed with the new work function $w^*_{v-\Delta}(.)$):

- Initial step $i = 0$: $w^*_{v-\Delta}(v) \geq w_{v-\Delta}(v)$ by definition of w^*.

- Induction assumption at step i:

$$w^*_{v-\Delta+i}(v) \geq w_{v-\Delta+i}(v) \tag{8.18}$$

- Let us prove the induction property at step $i + 1$, i.e., that $w^*_{v-\Delta+i+1}(v) \geq w_{v-\Delta+i+1}(v)$. Let $h := w^*_{v-\Delta+i}(v) - w_{v-\Delta+i}(v)$. We first have:

$$\pi^*(v - \Delta + i) = \frac{w^*_{v-\Delta+i}(v)}{\Delta - i}$$

  because, at any time $r < v$, we have $w^*_{v-\Delta+i}(r) = 0$ by construction.

For the original system, $\pi(v-\Delta+i) \geq \frac{w_{v-\Delta+i}(v)}{\Delta-i}$ because the maximum could be reached for some $r < v$. This yields:

$$\begin{aligned}
w_{v-\Delta+i}(v) - \pi(v-\Delta+i) &\leq w_{v-\Delta+i}(v) - \frac{w_{v-\Delta+i}}{\Delta-i} \\
&= w^*_{v-\Delta+i}(v) - h - \frac{w^*_{v-\Delta+i}(v) - h}{\Delta-i} \\
&= w^*_{v-\Delta+i}(v) - \frac{w^*_{v-\Delta+i}(v)}{\Delta-i} - \left(h - \frac{h}{\Delta-i}\right) \\
&\leq w^*_{v-\Delta+i}(v) - \frac{w^*_{v-\Delta+i}(v)}{\Delta-i} \\
&= w^*_{v-\Delta+i}(v) - \pi^*(v-\Delta+i). \qquad (8.19)
\end{aligned}$$

Furthermore, for each i, $w_{v-\Delta+i}(v)$ is the total amount of work present in the original system at time $v-\Delta+i$, because we have discarded all jobs with deadline larger than v in Step 1. This implies $\pi(v-\Delta+i) \leq w_{v-\Delta+i}(v)$, hence:

$$\big(w_{v-\Delta+i}(v) - \pi(v-\Delta+i)\big)_+ = w_{v-\Delta+i}(v) - \pi(v-\Delta+i). \qquad (8.20)$$

Putting Eqs. (8.19) and (8.20) together, since for all $k \leq t$, $A^*(k,v) = A(k,v)$, we get:

$$\begin{aligned}
w^*_{v-\Delta+i+1}(v) &= \big(w^*_{v-\Delta+i}(v) - \pi^*(v-\Delta+i)\big)_+ + A^*(v-\Delta+i+1,v) \\
&\geq \big(w_{v-\Delta+i}(v) - \pi(v-\Delta+i)\big) + A(v-\Delta+i+1,v) \\
&= \big(w_{v-\Delta+i}(v) - \pi(v-\Delta+i)\big)_+ + A(v-\Delta+i+1,v) \\
&= w_{v-\Delta+i+1}(v),
\end{aligned}$$

which is the property we wanted to prove at step $i+1$. This finishes Step 2.

Step 3: In the star system (work function $w^*_{v-\Delta}(.)$), the speeds used by (OA) at times $v-\Delta$ to $t-1$ are respectively:

$$\begin{aligned}
\pi^*(v-\Delta) &= \frac{w^*_{v-\Delta}(v)}{\Delta} \\
\pi^*(v-\Delta+1) &= \frac{w^*_{v-\Delta}(v)}{\Delta} + \frac{A(v-\Delta+1,v)}{\Delta-1} \\
\pi^*(v-\Delta+2) &= \frac{w^*_{v-\Delta}(v)}{\Delta} + \frac{A(v-\Delta+1,v)}{\Delta-1} + \frac{A(v-\Delta+2,v)}{\Delta-2} \\
&\vdots \\
\pi^*(t-1) &= \frac{w^*_{v-\Delta}(v)}{\Delta} + \frac{A(v-\Delta+1,v)}{\Delta-1} + \frac{A(v-\Delta+2,v)}{\Delta-2} + \ldots + \frac{A(t-1,v)}{v-t+1}.
\end{aligned}$$

We introduce the variable $u = v - t$ and compute the sum of all speeds:

$$\sum_{i=v-\Delta}^{t-1} \pi^*(i) = \frac{(\Delta-u)w^*_{v-\Delta}(v)}{\Delta} + \frac{(\Delta-u-1)A(v-\Delta+1,v)}{\Delta-1} + \ldots + \frac{A(t-1,v)}{u+1}. \qquad (8.21)$$

We then compute the sum of Eqs. (8.15) to (8.17), in the case of the star system. Note that the speeds $\pi^*(i)$ never become larger than the work $w^*(i)$, so the $\max$ operator is never "active" and can be removed:

$$w_t^*(v) = w_{v-\Delta}^*(v) - \sum_{i=v-\Delta}^{t-1} \pi^*(i) + \sum_{i=v-\Delta+1}^{t} A(i,v). \tag{8.22}$$

By replacing in Eq. (8.22) the sum of the speeds (Eq. (8.21)), we obtain the remaining work at $v = t+u$:

$$w_t^*(v) \quad = \quad \frac{uw_{v-\Delta}^*(v)}{\Delta} + \frac{uA(v-\Delta+1,v)}{\Delta-1} + \dots + \frac{uA(t-1,v)}{u+1} + A(t,v). \tag{8.23}$$

Since $w_{v-\Delta}^*(v) \le C\Delta$ (see Eq. (8.8)) and $A(k,v) \le C$ for all $k \le t$, we obtain an upper bound on $w_t^*(t+u)$:

$$w_t^*(t+u) \quad \le \quad \frac{uC\Delta}{\Delta} + \frac{uC}{\Delta-1} + \dots + \frac{uC}{u+1} + C. \tag{8.24}$$

This finishes Step 3 and provides a bound on $w_t(t+u)$, for all t, because $w_t(t+u) \le w_t^*(t+u)$.

To find an upper bound on $\pi^{(\mathrm{OA})}(t)$, we divide by u:

$$\frac{w_t(t+u)}{u} \le \frac{C\Delta}{\Delta} + \frac{C}{\Delta-1} + \dots + \frac{C}{u+1} + \frac{C}{u}.$$

The bound on the right hand side of Eq. (8.5.2) is maximal when $u=1$. We therefore get an upper bound on $\pi^{(\mathrm{OA})}(t)$, denoted $\overline{w}_1$:

$$\pi^{(\mathrm{OA})}(t) \le \underbrace{C + \frac{C}{\Delta-1} + \dots + \frac{C}{2} + C}_{\overline{w}_1}. \tag{8.25}$$

The star remaining function w^* (as displayed in Figure 8.1) is not reachable under (OA). However, one can construct a remaining work function that is asymptotically arbitrarily close to it. This construction is illustrated in Figure 8.2. First, jobs of size C and relative deadline Δ arrive at each slot during n time slots. When n grows to infinity, the speed selected by (OA) approaches C and the remaining work approaches the black staircase displayed in Figure 8.2 (see Lemma 8.1 below). Then, jobs of size C and absolute deadline $\Delta+n$ arrive at all time slots from $n+1$ to $n+\Delta-1$ (job arrivals are represented alternatively in blue and red in Figure 8.2). In that case, we will show that $w_{n+\Delta-1}^{(\mathrm{OA})}(n+1)$ will approach $\overline{w}_1$ as n goes to infinity.

Lemma 8.1. *If the sequence of jobs is such that at each time n a job arrives with size C and relative deadline Δ, then:*

- *The speeds $\pi^{(\mathrm{OA})}(n)$ increase and converge towards C when n goes to infinity;*

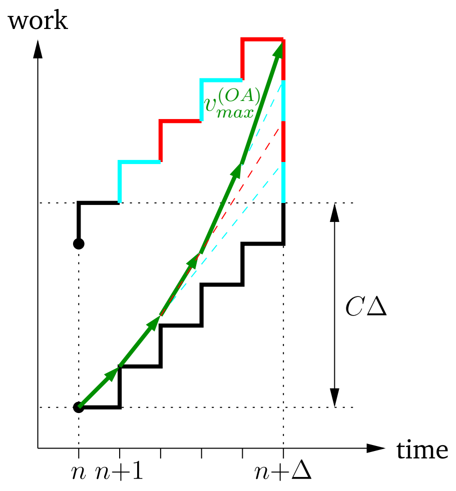


Figure 8.2.: Asymptotic worst case state, for $C = 1$ and $\Delta = 5$. The staircase black curve represents the remaining work function reached asymptotically while the coloured parts (blue and red segments represent one job at each time slot with identical WCET C and identical absolute deadline) lead to the maximal $w^{(\mathrm{OA})}_{n+\Delta-1}(n+\Delta)$. The green arrows represent the quantities of work executed by the processor under (OA).

- *The remaining work function converges towards the function $w_n(.)$ such that $\forall i \le \Delta$, $w_n(n+i) = iC$.*

Proof. We show by induction on n that $w_n(n+\Delta-1) \le C(\Delta-1)$ and that $\pi(n) = w_n(n+\Delta)/\Delta$.

- Initial step $n = 0$: a single job has arrived, with size C and deadline Δ. Therefore, $w_0(\Delta-1) = 0 \le C(\Delta-1)$ and $\pi(0) = w_0(\Delta)/\Delta = C/\Delta$.

- Induction assumption at step n:

$$w_n(n+\Delta-1) \le C(\Delta-1) \quad \wedge \quad \pi(n) = w_n(n+\Delta)/\Delta. \tag{8.26}$$

- Let us prove the induction property at step $n+1$. We have:

$$\begin{aligned} w_{n+1}(n+\Delta) &= (w_n(n+\Delta) - \pi(n))_+ + A(n+1, n+\Delta) \\ &= w_n(n+\Delta) - \frac{w_n(n+\Delta)}{\Delta} + 0 \\ &= w_n(n+\Delta)\frac{\Delta-1}{\Delta}. \end{aligned}$$

  Since $w_n(n+\Delta)$ is always smaller than $C\Delta$ (see Eq. (8.8)), it follows that:

$$w_{n+1}(n+\Delta) \le C(\Delta-1). \tag{8.27}$$

Let us now compute the speed π at time $n+1$. Since the speed at time n is $\pi(n) = \max_v \frac{w_n(n+v)}{v} = \frac{w_n(n+\Delta)}{\Delta}$, which is not reached for $v=1$, then $\pi(n+1) = \max(\pi(n), w_{n+1}(n+\Delta+1)/\Delta)$. Replacing $\pi(n)$ by its value from the induction hypothesis yields:

$$\begin{aligned}\pi(n+1) &= \max\left(\frac{w_n(n+\Delta)}{\Delta}, \frac{w_{n+1}(n+\Delta+1)}{\Delta}\right) \\ &= \frac{1}{\Delta}\max\left(w_n(n+\Delta), w_{n+1}(n+\Delta+1)\right). \end{aligned} \tag{8.28}$$

Since the job that arrives at time $n+\Delta+1$ is of size C, the second term of the $\max$ is:

$$w_{n+1}(n+\Delta+1) = w_{n+1}(n+\Delta) + C = w_n(n+\Delta)\frac{\Delta-1}{\Delta} + C.$$

Using the induction assumption. It follows that:

$$w_{n+1}(n+\Delta+1) = w_n(n+\Delta) + \left(C - \frac{w_n(n+\Delta)}{\Delta}\right).$$

We again use the fact that $w_n(n+\Delta) \leq C\Delta$ (see Eq. (8.8)) to conclude that the second term of the $\max$ is always larger than the first term, giving:

$$\pi(n+1) = \frac{w_{n+1}(n+\Delta+1)}{\Delta}. \tag{8.29}$$

This ends the induction and shows as a byproduct that $\pi(n+1) \geq \pi(n)$.

Now, since $\pi(n)$ is increasing, it converges to some value $L \leq \infty$. Since for all n, $0 \leq w_n(n+\Delta) \leq c\Delta$, and since $w_n(n+\Delta) = \sum_{i=0}^{n} C - \sum_{i=0}^{n-1} \pi(i)$ is equivalent to $n(C-L)$ when n grows, then $L = C$.

As for the second part of the Lemma, it follows from inspecting Eq. (8.28). The fact that $\pi(n+1) - \pi(n)$ goes to 0, implies that $w_{n+1}(n+\Delta)$ goes to $C(\Delta-1)$. This implies that $w_{n+1}(n+1+i)$ goes to Ci, for all $0 \leq i \leq \Delta$. This concludes the proof of Lemma 8.1. □

In the following we use the following notation: $x_n \approx y_n$ if $|x_n - y_n| \leq \varepsilon$, for some $\varepsilon > 0$ arbitrarily small.

Let us now resume the proof of Theorem 8.1. Consider the following job sequence: First n jobs arrive, with release times $1, 2, \ldots n$, size C and *relative* deadline Δ, the next jobs arrive at times $n+1, n+2, \ldots n+\Delta-1$ with size C and *absolute* deadline $n+\Delta$. We assume that n is large enough so that using Lemma 8.1, $w_n(n+i) \approx Ci$ for all $i \leq \Delta$ and $\pi^{(\mathrm{OA})}(n) \approx C$ (see Figure 8.2).

By construction of the job sequence,

- at time $n+1$, $\pi^{(\mathrm{OA})}(n+1) \approx \dfrac{C\Delta}{\Delta-1}$;

- more generally, for $1 \leq k < \Delta$, we have on the one hand:

$$\begin{aligned}
&(\Delta - k)\pi^{(\mathrm{OA})}(n+k) \approx (\Delta + k - 1)C - \sum_{j=1}^{k-1} \pi^{(\mathrm{OA})}(n+j) \\
\iff\ &(\Delta-k-1)\pi^{(\mathrm{OA})}(n+k) \approx (\Delta+k-1)C - \sum_{j=1}^{k-1}\pi^{(\mathrm{OA})}(n+j) - \pi^{(\mathrm{OA})}(n+k) \\
\iff\ &(\Delta - k - 1)\pi^{(\mathrm{OA})}(n+k) \approx (\Delta + k - 1)C - \sum_{j=1}^{k} \pi^{(\mathrm{OA})}(n+j) \qquad (8.30)
\end{aligned}$$

and on the other hand:

$$(\Delta - k - 1)\pi^{(\mathrm{OA})}(n+k+1) \approx (\Delta + k)C - \sum_{j=1}^{k} \pi^{(\mathrm{OA})}(n+j). \qquad (8.31)$$

By subtracting Eq. (8.30) to Eq. (8.31), we obtain:

$$\begin{aligned}
&(\Delta - k - 1)\left(\pi^{(\mathrm{OA})}(n+k+1) - \pi^{(\mathrm{OA})}(n+k)\right) \approx C \\
\iff\ &\pi^{(\mathrm{OA})}(n+k+1) \approx \pi^{(\mathrm{OA})}(n+k) + \frac{C}{\Delta - k - 1}. \qquad (8.32)
\end{aligned}$$

By applying iteratively Eq. (8.32) from $n+k+1$ down to $n+1$, we obtain for all $k \geq 1$:

$$\begin{aligned}
\pi^{(\mathrm{OA})}(n+k+1) &\approx \pi^{(\mathrm{OA})}(n+1) + C\sum_{j=2}^{k+1} \frac{1}{\Delta - j} \\
&\approx \pi^{(\mathrm{OA})}(n+1) + C(h_{\Delta-2} - h_{\Delta-k-2})
\end{aligned}$$

where h_n is the n-th harmonic number: $h_n = \sum_{i=1}^{n} 1/i$ and $h_0 = 0$. Therefore $\pi^{(\mathrm{OA})}(n + \Delta - 1)$ has the following asymptotic value:

$$C\left(\frac{\Delta}{\Delta - 1} + h_{\Delta-2}\right) = C\left(1 + h_{\Delta-1}\right) = \overline{w}_1,$$

by using $\pi^{(\mathrm{OA})}(n+1) \approx C\frac{\Delta}{\Delta-1}$.

To conclude, the (OA) policy may use a speed arbitrarily close to its upper bound, $\overline{w}_1$. Therefore, it is feasible if and only if

$$s_{\max} \geq \overline{w}_1 = C\left(1 + h_{\Delta-1}\right). \qquad (8.33)$$

This concludes the proof of Theorem 8.1 in the case $\mathcal{T} = \mathbb{N}$. □

♦ **The speed decision instants are real numbers: $\mathcal{T} = \mathbb{R}$.**

We will prove that, when (OA) is given the opportunity to change the speed at any time $t \in \mathbb{R}$, the speed chosen at any real time t is the same as the speed chosen at the previous integer instant.

Let us denote by $w_k^{\mathbb{N}}$ the remaining work under integer decision instants, and $w_k^{\mathbb{R}}$ the remaining work under real decision instants. We will prove by induction on k that for any integer k, $w_k^{\mathbb{R}} = w_k^{\mathbb{N}}$.

For the sake of simplicity, let us denote as π instead of $\pi^{\mathbb{R}}$ the (OA) speed function when the speeds can change at any real instant.

For all $k \in \mathbb{N}$, for all t such that $k < t < k+1$ and all $v \geq t$, we recall the formula giving the remaining work, when the current time is t (see Eq. (8.7)):

$$w_t^{\mathbb{R}}(v) = \left(w_k^{\mathbb{R}}(v) - \int_k^t \pi(x)\mathrm{d}x\right)_+ + A(t,v) = \left(w_k^{\mathbb{R}}(v) - \int_k^t \pi(x)\mathrm{d}x\right)_+ \tag{8.34}$$

and according to Def. 8.6 of the (OA) policy, at each time $t \in \mathbb{R}$ we have:

$$\pi(t) \quad = \quad \max_{v>t}\left(\frac{w_t^{\mathbb{R}}(v)}{v-t}\right). \tag{8.35}$$

Combining Eqs. (8.34) and (8.35) yields:

$$\pi(t) \quad = \quad \max_{v>t}\frac{\left(w_k^{\mathbb{R}}(v) - \int_k^t \pi(x)\mathrm{d}x\right)_+}{v-t}. \tag{8.36}$$

The $(.)_+$ operator can be removed in Eq. (8.36) because, for $v = k+\Delta$, $w_k^{\mathbb{R}}(k+\Delta) \geq \int_k^t \pi(x)dx$ (see Remark 8.2). It follows that:

$$\pi(t) \quad = \quad \max_{v>t}\frac{\left(w_k^{\mathbb{R}}(v) - \int_k^t \pi(x)\mathrm{d}x\right)}{v-t}. \tag{8.37}$$

Let us now prove by induction on k that $\forall k \in \mathbb{N}$, $w_k^{\mathbb{R}} = w_k^{\mathbb{N}}$.

- Initial step $k=0$: only the first job J_1 may have arrived at time 0. Therefore, for all $v \geq 0$, $w_0^{\mathbb{R}}(v) = w_0^{\mathbb{N}}(v) = c_1$ if $r_1 = 0, d_1 \leq v$ and $w_0^{\mathbb{R}}(v) = w_0^{\mathbb{N}}(v) = 0$ otherwise.

- Induction assumption at step k:
$$w_k^{\mathbb{R}} = w_k^{\mathbb{N}}. \tag{8.38}$$

- Now consider $t \in \mathbb{R}$ such that $k < t < k+1$. We first prove that $\pi(t) = \pi(k)$.

  We define m as
$$m := \operatorname*{argmax}_v \frac{w_k^{\mathbb{R}}(v)}{v-k}. \tag{8.39}$$

  This means that
$$\pi(k) = \frac{w_k^{\mathbb{R}}(m)}{m-k}. \tag{8.40}$$

Now let us check whether a constant speed on $[k, k+1)$ — *i.e.* $\pi(t) = \pi(k), \forall t \in [k, k+1)$ — can be a solution of Eq. (8.37), the integral equation defining π. With a constant speed, the numerator in Eq. (8.37) becomes:

$$w_t^{\mathbb{R}}(v) = w_k^{\mathbb{R}}(v) - \pi(k)(t-k). \tag{8.41}$$

By using the value of $\pi(k)$ from Eq. (8.40), we obtain:

$$\frac{w_t^{\mathbb{R}}(v)}{v-t} = \frac{w_k^{\mathbb{R}}(v)(m-k) - w_k^{\mathbb{R}}(m)(t-k)}{(v-t)(m-k)}. \tag{8.42}$$

First, the particular case where $v = m$ leads to:

$$\frac{w_t^{\mathbb{R}}(m)}{m-t} = \frac{w_k^{\mathbb{R}}(m)(m-k) - w_k^{\mathbb{R}}(m)(t-k)}{(m-t)(m-k)} = \frac{w_k^{\mathbb{R}}(m)}{m-k} = \pi(k). \tag{8.43}$$

Second, we show that this particular case is also the maximal value for $w_t^{\mathbb{R}}(v)/(v-t)$. Eq. (8.40) implies that: $\frac{w_k^{\mathbb{R}}(v)}{v-k} \leq \frac{w_k^{\mathbb{R}}(m)}{m-k}$. Together with Eq. (8.42), this yields:

$$\begin{aligned}
\forall v \in \mathbb{R}, \quad \frac{w_t^{\mathbb{R}}(v)}{v-t} &= \frac{w_k^{\mathbb{R}}(v)(m-k) - w_k^{\mathbb{R}}(m)(t-k)}{(v-t)(m-k)} \\
&\leq \frac{w_k^{\mathbb{R}}(m)(v-k) - w_k^{\mathbb{R}}(m)(t-k)}{(v-t)(m-k)} \\
&= \frac{w_k^{\mathbb{R}}(m)(v-t)}{(v-t)(m-k)} \\
&= \pi(k).
\end{aligned} \tag{8.44}$$

By Appendix (8.11), the solution of Eq. (8.37) is unique. Therefore the solution of this equation is:

$$\forall t \in [k, k+1) \quad \pi(t) = \pi(k). \tag{8.45}$$

Since the speed is constant between two integer time steps, and since, by the induction assumption (8.38), $w_k^{\mathbb{R}} = w_k^{\mathbb{N}}$, we thus have $w_{k+1}^{\mathbb{R}} = w_{k+1}^{\mathbb{N}}$. This concludes the induction proof.

This induction proof also shows that the speed decision are the same for integer and real decision instant. This implies that the behaviour of (OA) with real decision instants is the same as the behaviour of (OA) with integer decision instants. Therefore the feasibility condition is the same. This concludes the proof of Theorem 8.1. $\square$

8.6 Feasibility of the Average Rate speed policy (AVR)

8.6.1 Definition of (AVR) [YDS95]

(AVR) is defined in [YDS95] as follows:

Definition 8.7 (AVerage Rate (AVR)). *At each time $t \in \mathcal{T}$, the job that has the earliest deadline is executed at speed:*

$$\pi^{(\mathrm{AVR})}(t) = \sum_{i \in \mathcal{A}(t)} \frac{c_i}{D_i - r_i} \tag{8.46}$$

where $\mathcal{A}(t)$ is the set of active *jobs at time t,* i.e. *jobs $J_i = (r_i, c_i, D_i)$ such that $r_i \le t < D_i$.*

Notice that the processor speed $\pi^{(\mathrm{AVR})}(t)$ is independent of the previous speeds used by the processor. In contrast, (OA) chooses at time t a speed that, through $w^{(\mathrm{OA})}$, depends on the previous speeds used by the processor.

Let us apply (AVR) policy on the example displayed in Section (8.5), where we consider the same 3 jobs:

- $J_1 = (r_1 = 0, c_1 = 1, d_1 = 4)$
- $J_2 = (r_2 = 3, c_2 = 4, d_2 = 6)$
- $J_3 = (r_3 = 3, c_3 = 1, d_3 = 8)$

The three jobs are active at time 3, thus using Eq. (8.46) yields:

$$\pi^{(\mathrm{AVR})}(3) = \frac{c_1}{d_1 - r_1} + \frac{c_2}{d_2 - r_2} + \frac{c_3}{d_3 - r_3} = \frac{1}{4-0} + \frac{4}{6-3} + \frac{1}{8-3}$$

Therefore $\pi^{(\mathrm{AVR})}(3) = \frac{107}{60}$.

We note that the speed chosen at time 3 by (AVR) is greater than the one chosen by (OA). However, in the next section, we will show that the maximal speed required by (AVR) for feasibility is smaller than the maximal speed required by (OA) and determined in Section 8.5.

8.6.2 Feasibility analysis

Theorem 8.2 establishes the condition on $s_{\max}$ that insures the feasibility of (AVR).

Theorem 8.2. (AVR) *is feasible* $\iff s_{\max} \ge Ch_{\Delta}$.

Proof. We distinguish the cases where the decision instants are integer and real numbers.

♦ **The decision instants are integer numbers:** $\mathcal{T} = \mathbb{N}$.

According to Prop. 8.1, the (AVR) feasibility proof is split in two different parts.

▶ The first part consists in showing that all jobs are executed before their deadlines, *i.e.* $\pi^{(\text{AVR})}(t) \geq w_t^{(\text{AVR})}(t+1)$.

Let us focus on one job $J_i = (r_i, c_i, D_i)$. Under (AVR), one can consider that the processor dedicates a fraction of its computing power to execute a quantity of work equal to $\frac{c_i}{d_i}$ per time unit from r_i to $r_i + d_i - 1$, for job J_i only. So at time $r_i + d_i$, the job J_i is totally executed by the processor, hence before its deadline. Since this reasoning is valid for all jobs, all jobs are executed before their deadline under (AVR) as long as $s_{\max}$ is large enough.

Therefore, the feasibility equation (8.10) can be simplified and written as a condition on $s_{\max}$:

$$\text{(AVR) is feasible} \iff \forall t \in \mathcal{T},\ s_{\max} \geq \pi^{(\text{AVR})}(t). \tag{8.47}$$

▶ Let us now compute the minimal value of $s_{\max}$ such that (AVR) is feasible, by building a worst-case scenario:

- By definition of (AVR), there is no influence of the work already executed on the value of the current speed. We therefore focus on the currently active jobs.
- $\pi^{(\text{AVR})}(t)$ increases with the size of each job, so we consider jobs of maximal size, namely C.
- $\pi^{(\text{AVR})}(t)$ increases with the number of active jobs, so our worst-case scenario involves the maximal possible number of active jobs, namely Δ (because only one job of size C can arrive at each time step, with a deadline not larger than Δ).
- $\pi^{(\text{AVR})}(t)$ increases when the deadline of the jobs are small, so we consider jobs with the smallest possible deadline, namely $t + 1$.

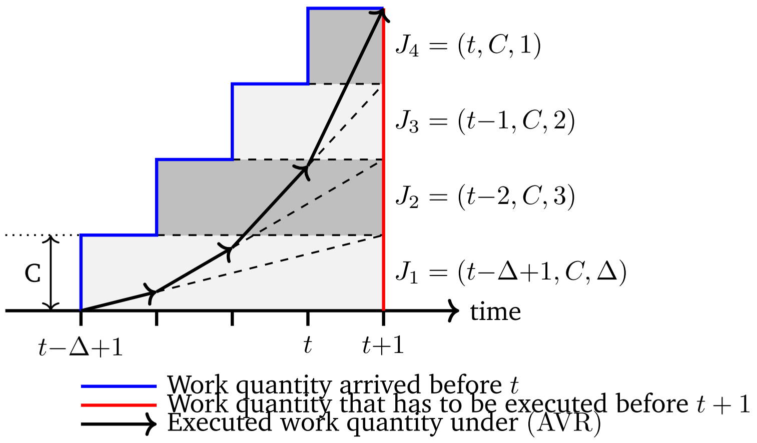

$J_4 = (t, C, 1)$
$J_3 = (t{-}1, C, 2)$
$J_2 = (t{-}2, C, 3)$
$J_1 = (t{-}\Delta{+}1, C, \Delta)$
C
time
$t{-}\Delta{+}1$
t
$t{+}1$
Work quantity arrived before t
Work quantity that has to be executed before $t+1$
Executed work quantity under (AVR)

Figure 8.3.: Worst-case scenario for (AVR) when $\Delta = 4$.

In this worst-case scenario (illustrated in Fig. 8.3 when $\Delta = 4$), the speed $\pi^{(\mathrm{AVR})}(t)$ is maximal at time t and is the sum of the average speed of each active job J_i executed separately:

$$\pi^{(\mathrm{AVR})}(t) = \sum_{i=1}^{\Delta} \frac{C}{i} = Ch_{\Delta}.$$

It follows that the feasibility condition for (AVR) is:

$$Ch_{\Delta} \leq s_{\max}.$$

This worst-case scenario allows us to determine the maximal processor speed $s_{\max}$ under which the (AVR) policy can schedule any sequence of jobs without missing a deadline. If we suppose that $s_{\max} \leq Ch_{\Delta}$, then there exists a job configuration on which (AVR) is not feasible, as shown in Fig 8.3. Therefore Theorem 8.2 is proved.

♦ **The speed decision instants are real numbers: $\mathcal{T} = \mathbb{R}$.**
By definition, $\pi^{(\mathrm{AVR})}(t)$ only depends on the set of active jobs, satisfying $r_i \leq t < D_i$. Since r_i and D_i are integer numbers, the set of active jobs is the same for t and for $\lfloor t \rfloor$. As for the previous policy, allowing real decision instants for (AVR) does not change the chosen speeds. We thus have the same feasibility condition for the integer and real decision instants, which is $s_{\max} \geq Ch_{\Delta}$. □

8.7 Feasibility of the Bansal, Kimbrel, Pruhs speed policy (BKP)

8.7.1 Definition of (BKP) [BKP07]

Definition 8.8 (Contributing work). *For any t, t_1, and t_2 in $\mathbb{R}$such that $t_1 \leq t \leq t_2$, $u(t, t_1, t_2)$ is the amount of work arrived after t_1 and before t, the deadline of which is less than t_2.*

According to Def. 8.8, any job $J_i = (r_i, c_i, D_i)$ contributing to $u(t, t_1, t_2)$ must satisfy $t_1 \leq r_i \leq t$ and $D_i \leq t_2$.

Definition 8.9 (Bansal, Kimbrel, Pruhs policy (BKP)). *At each time t, the job that has the earliest deadline is executed at speed:*

$$\pi^{(\mathrm{BKP})}(t) = \max_{t_2 > t} \left\{ \frac{u(t, et - (e-1)t_2, t_2)}{t_2 - t} \right\}. \tag{8.48}$$

Remark 8.3. (BKP) *was designed to improve the competitive ratio of* (OA)*, from α^{α} for* (OA) *to* $2(\frac{\alpha e}{\alpha - 1})^{\alpha}$ *for* (BKP)*, when the power dissipated by the processor at speed s is s^{α} [BKP07].*

Let us apply the policy (BKP) on the simple example displayed in Sections 8.5 and 8.6. We recall the 3 jobs:

- $J_1 = (r_1 = 0, c_1 = 1, d_1 = 4)$,
- $J_2 = (r_2 = 3, c_2 = 4, d_2 = 6)$,
- $J_3 = (r_3 = 3, c_3 = 1, d_3 = 8)$.

For computing the max in Eq. (8.48) at time 3, let us examine four possible cases:

1. $t_2 > d_3$: In that case, the 3 jobs are present at time u, hence:
$$\frac{u(3, 3e - (e-1)t_2, t_2)}{t_2 - 3} \leq \frac{c_1 + c_2 + c_3}{d_3 - 3} = \frac{6}{5}.$$
2. $d_2 < t_2 < d_3$: Because of the deadlines, in the best case, only 2 jobs J_1 and J_2 are present at time u, hence:
$$\frac{u(3, 3e - (e-1)t_2, t_2)}{t_2 - 3} \leq \frac{c_1 + c_2}{d_2 - 3} = \frac{5}{3}.$$
3. $t_2 = d_2$: The 2 jobs J_1 and J_2 are present at time u, hence:
$$\frac{u(3, 3e - (e-1)t_2, t_2)}{t_2 - 3} = \frac{c_1 + c_2}{d_2 - 3} = \frac{5}{3}.$$
4. $t_2 < d_2$: Only job J_1 can be present at time u, hence:
$$\frac{u(3, 3e - (e-1)t_2, t_2)}{t_2 - 3} \leq \frac{c_1}{d_1 - 3} = 1.$$

As a consequence, we obtain:
$$\pi^{(\mathrm{BKP})}(3) = \frac{c_1 + c_2}{d_2 - 3} = \frac{5}{3}.$$

The following table summarizes the numerical values computed by the three speed policies (OA), (AVR), and (BKP) at time 3 and for the chosen example with three jobs.

(OA)	(AVR)	(BKP)
17/12	107/60	5/3

We therefore have the following inequality:
$$\pi^{(\mathrm{OA})}(3) \leq \pi^{(\mathrm{BKP})}(3) \leq \pi^{(\mathrm{AVR})}(3).$$

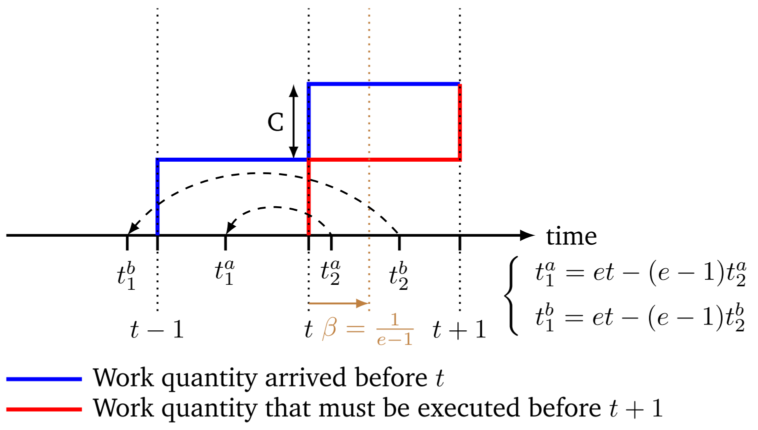
In the following, we will show that even if the behavior of (BKP) looks like a compromise between (OA) and (AVR), the feasibility condition of (BKP) is much better than both.

8.7.2 Feasibility analysis of (BKP) with $\mathcal{T} = \mathbb{N}$

Theorem 8.3. (BKP) *is feasible with* $\mathcal{T} = \mathbb{N} \iff s_{\max} \geq \frac{3}{2}(e-1)C$.

Proof. From Theorem 5 in [BKP07], (BKP) completes all the jobs by their deadlines. As a consequence, $\forall t \in \mathbb{R}$ (and hence in $\mathbb{N}$), we have $\pi^{(\mathrm{BKP})}(t) \geq w_t^{(\mathrm{BKP})}(t+1)$. Therefore, the feasibility equation, Eq. (8.10), can be simplified and rewritten as a condition only on $s_{\max}$:

$$(\mathrm{BKP}) \text{ is feasible with } \mathcal{T} = \mathbb{N} \iff \forall t \in \mathbb{N},\, s_{\max} \geq \pi^{(\mathrm{BKP})}(t). \tag{8.49}$$



C
time
t_1^b
t_1^a
t_2^a
t_2^b
$t-1$
t
$\beta = \frac{1}{e-1}$
$t+1$
$t_1^a = et - (e-1)t_2^a$
$t_1^b = et - (e-1)t_2^b$
Work quantity arrived before t
Work quantity that must be executed before $t+1$

Figure 8.4.: (BKP) with integer speed decision instants ($t \in \mathbb{N}$) and Case 1 ($t_2 < t+1$). Let $t_2^i \in (t, t+1]$ with $i \in \{a, b\}$ to illustrate the two sub-cases. Before $t+\beta$ (sub-case $i = a$), no jobs are taken into account in the speed computation, so $s_{\max}^{(\mathrm{BKP})} = 0$. After this threshold (sub-case $i = b$), $s_{\max}^{(\mathrm{BKP})}$ can be non null because we take potentially into account the job arriving at $t-1$ and ending at t. This job is at worst of size C. The two black arrows illustrate the position of t_1^i with respect to that of t_2^i.

In order to prove Condition (8.49), we will find an upper and a lower bound for the maximal speed of (BKP). To find an upper bound on $s_{\max}^{(\mathrm{BKP})}$, we have to determine an upper bound on $u(t, t_1, t_2)$. Let $t \in \mathbb{N}$. We split the analysis in two cases:

▶ **Case 1:** We consider the case where $t_2 - t < 1$. We are faced with two subcases:

- Either $t_1 = et - (e-1)t_2 > t-1$. In that subcase, no job can arrive after t_1 with a deadline smaller than t_2. Therefore $u(t, t_1, t_2) = 0$, and $\pi^{(\mathrm{BKP})}(t) = 0$. This subcase is illustrated in Fig. 8.4 by the tuple (t, t_1^a, t_2^a).

- Or $t_1 = et - (e-1)t_2 \leq t-1$. Here, potentially, one job can arrive at $t-1$ and end at t. We introduce the variable $\beta \in \mathbb{R}$ such that $t_2 = t + \beta$ and $t_1 \leq t-1$. This limit case (the earliest t_2, under these conditions, such that one job can be taken into account in the (BKP) speed computation) leads to:

$$\begin{aligned} & t_1 = t-1 \\ \iff \quad & et-(e-1)(t+\beta) = t-1 \\ \iff \quad & \beta = \frac{1}{e-1}. \end{aligned}$$

Therefore the maximal value for $u(t,t_1,t_2)$ is $\frac{C}{\beta}$ and is reached for $t_2 = t+\beta$. Note that $\frac{C}{\beta}$ is independent of t. This subcase is illustrated in Fig. 8.4 by the tuple (t, t_1^b, t_2^b).

▶ **Case 2:** We consider the case where $t_2 \geq t+1$. In this case the contributing work is bounded by:

$$u(t,t_1,t_2) \leq C\lfloor t-t_1+1 \rfloor. \tag{8.50}$$

because, even when $t = t_1$, one job can arrive at t and be taken into account by (BKP) (hence the "$+1$"). It follows that the speed computed by (BKP) is:

$$\begin{aligned} \pi^{(\mathrm{BKP})}(t) \quad & \leq \quad C \max_{t_2>t}\left\{\frac{\lfloor t-et+(e-1)t_2+1 \rfloor}{t_2-t}\right\} \\ & \leq \quad C \max_{t_2>t}\left\{\frac{\lfloor (e-1)(t_2-t)+1 \rfloor}{t_2-t}\right\}. \end{aligned} \tag{8.51}$$

To reason about Eq. (8.51), we introduce the variable $\gamma \in \mathbb{R}_+$, such that $t_2 = t+1+\gamma$. Accordingly, $\pi^{(\mathrm{BKP})}$ depends only on γ:

$$\pi^{(\mathrm{BKP})}(\gamma) \quad \leq \quad C \max_{\gamma\in\mathbb{R}_+}\left\{\frac{\lfloor (e-1)(1+\gamma)+1 \rfloor}{1+\gamma}\right\}. \tag{8.52}$$

Because of the floor operator, $(e-1)(1+\gamma)+1$ must be in $\mathbb{N}$for the fraction $\frac{\lfloor (e-1)(1+\gamma)+1 \rfloor}{1+\gamma}$ to be maximized, and since $e-1$ is irrational, there must exist $k \in \mathbb{N}$ such that:

$$1+\gamma = \frac{k}{e-1}. \tag{8.53}$$

It follows that:

$$\frac{\lfloor (e-1)(1+\gamma)+1 \rfloor}{1+\gamma} = \frac{(e-1)(1+\gamma)+1}{1+\gamma} = e-1+\frac{1}{1+\gamma}. \tag{8.54}$$

Now, since γ is positive, we have $1+\gamma \geq 1$, so:

$$\frac{k}{e-1} \geq 1 \iff k \geq e-1. \tag{8.55}$$

The function $\gamma \mapsto e-1+\frac{1}{1+\gamma}$ is decreasing and γ has to satisfy the condition of the Inequality (8.55). Therefore the maximum of Eq. (8.52) is reached for the smallest $k \in \mathbb{N}$ (*i.e.* the smallest possible γ) such that Inequality (8.55) is satisfied:

$$k = \min_{j \in \mathbb{N}} \{j \geq e-1\} = \lceil e-1 \rceil. \tag{8.56}$$

Finally, by replacing k by its value in Eq. (8.53), we obtain:

$$\gamma = \frac{\lceil e-1 \rceil}{e-1} - 1 = \frac{3-e}{e-1} \simeq 0.16. \tag{8.57}$$

From Eqs. (8.52), (8.54), and (8.57), it follows that:

$$\pi^{(\mathrm{BKP})}(t) \quad \leq \quad C\left(e-1+\frac{e-1}{2}\right) = \frac{3}{2}(e-1)C \simeq 2.577\,C. \tag{8.58}$$

Putting Case 1 and Case 2 together, we obtain:

$$\begin{aligned}
\forall t, \pi^{(\mathrm{BKP})}(t) \quad &= \quad \max\left\{ \max_{t_2 \geq t+1} \left\{ \frac{u(t, et-(e-1)t_2, t_2)}{t_2 - t} \right\}, \right. \\
&\qquad\qquad \left. \max_{t+1 > t_2 > t} \left\{ \frac{u(t, et-(e-1)t_2, t_2)}{t_2 - t} \right\} \right\} \\
&\leq \quad \max\left\{ \max_{\gamma \in \mathbb{R}_+} \left\{ \left((e-1) + \frac{1}{1+\gamma} \right) C \right\}, (e-1)C \right\} \\
&\leq \quad \max_{\gamma \in \mathbb{R}_+} \left\{ \left((e-1) + \frac{1}{1+\gamma} \right) C \right\} \\
&\leq \quad \frac{3}{2}(e-1)C \simeq 2.577\,C.
\end{aligned} \tag{8.59}$$

We now want to establish a *lower bound* on the maximal speed of (BKP), by using Eq. (8.59). If we are in the particular case depicted in Figure 8.2 where $t = n + \Delta - 1$ and $t_2 = t + 1 + \gamma$, then we have $t_1 = et - (e-1)t_2 \in \mathbb{N}$ by definition of t_1 in Def. 8.9. Under these conditions and according to the previous computations done for the upper bound case, we have for this particular t:

$$\pi^{(\mathrm{BKP})}(t) = \frac{3}{2}(e-1)C. \tag{8.60}$$

Since the lower bound of Eq. (8.60) is equal to the upper bound of Eq. (8.59), we can conclude that:

$$(\mathrm{BKP}) \text{ is feasible} \iff s_{\max} \geq \frac{3}{2}(e-1)C.$$

□

8.7.3 Feasibility analysis of (BKP) with $\mathcal{T} = \mathbb{R}$

When the speed decision instants are real numbers, another feasibility condition holds for (BKP) policy, stated in Theorem 8.4.

Theorem 8.4. (BKP) *is feasible with* $\mathcal{T} = \mathbb{R} \iff s_{\max} \geq eC$.

Proof. Let us consider the same three following variables t, t_1, and t_2, as in the proof of Theorem 8.3. The only difference is the fact that t is in $\mathbb{R}$instead of $\mathbb{N}$.

Similarly to the proof of Theorem 8.3, we have to prove the following equivalence:

$$(\text{BKP}) \text{ is feasible with } \mathcal{T} = \mathbb{R} \iff \forall t \in \mathbb{R},\ s_{\max} \geq \pi^{(\text{BKP})}(t). \tag{8.61}$$

To do so, we will use the same method as in the previous proof, *i.e.* we determine an upper and a lower bound for the maximal speed of (BKP). To begin, we will find an *upper bound* on the maximal speed of (BKP). We introduce the variable $\beta \in \mathbb{R}$ such that $t_2 = t + \beta$. The set of jobs that are taken in consideration in (BKP) speed computation belongs to an interval of length $e\beta$, because:

$$t_2 - t_1 = t + \beta - et_1 + (e-1)(t+\beta) = e\beta.$$

This situation is depicted in Figure 8.5.

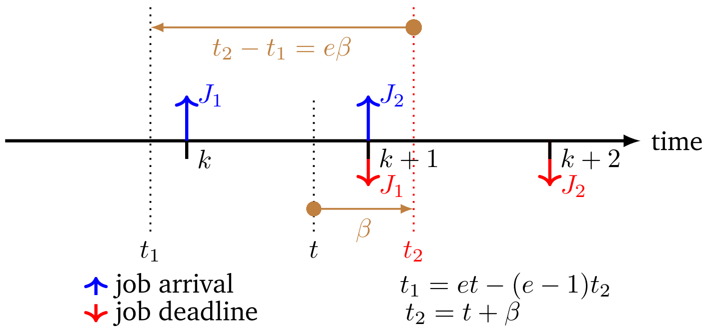


Figure 8.5.: Real speed decision instants, *i.e.* $t \in \mathbb{R}$ and $1 \leq t_2 - t_1 < 2$. Two jobs $J_1 = (k, C, 1)$ and $J_2 = (k+1, C, 1)$ are represented.

Let $n = \lfloor t_2 - t_1 \rfloor$, hence $n \leq t_2 - t_1 < n + 1$. Then at most $n + 1$ jobs can arrive in the $[t_1, t_2)$ interval and at most n of them can have a deadline before t_2, therefore $u(t, t_1, t_2) \leq nC$ so $\pi^{(\text{BKP})}(t) \leq \frac{nC}{n/e} = eC$. For all t in $\mathbb{R}$, an upper bound on (BKP) maximal speed is thus:

$$\pi_{\max}^{(\text{BKP})} \leq eC. \tag{8.62}$$

Now we consider the particular situation, where $\beta = 1/e$, $t_2 = 1$ and there is one job of size C with deadline 1 that arrives at time 0. In that case, $t_1 = 0$, $t = 1 - \frac{1}{e}$, and $t_2 = 1$. It follows that $u(t, t_1, t_2) = C$ and:

$$\pi^{(\mathrm{BKP})}(t) = \frac{C}{\beta} = eC. \tag{8.63}$$

As a conclusion, since the upper bound of Eq. (8.62) is reached (see Eq. (8.63)), we have:

$$(\mathrm{BKP}) \text{ is feasible} \iff eC \leq s_{\max}.$$

$\square$

8.8 Feasibility of the Markov Decision Process speed policy (MP)

This last policy shows that one can get the best of both worlds: An energy optimal policy whose feasibility region is maximal, at the price of statistical information about future jobs.

8.8.1 Definition of (MP) [GGP17]

In this section we assume that the job sequence $\{J_i\}_{i \in \mathbb{N}}$ is endowed with a probability distribution on (r_i, c_i, d_i). The precise values of the probabilities that a job is released at time r_i, is of size c_i, or has a relative deadline d_i are indeed important to compute the speed used at any time t by the online speed policy (MP), but they will not play a role in the feasibility analysis on (MP), as seen in the following.

To define (MP), we first introduce the *state* of the system at time t that gathers all the information useful to decide which speed to use at time t. Since all job features are integer numbers and the relative deadline is smaller than Δ, the current information at time t can be summarized in the vector $(w_t(t+1), w_t(t+2), \ldots, w_t(t+\Delta))$, which will be called the state at time t in the following, and denoted x_t.

Under this framework, we define the transition matrix $P_s(x, x')$, that gathers the probabilities to go from state x to state x' in one time step when the processor speed is s. The construction of this transition matrix requires to know the distribution of the release times, the sizes and the deadlines of future jobs. This knowledge may come from statistical analysis of the jobs in a training phase preceding the deployment of the speed policy in the system, or can even be learned online: the system adjusts its estimation of the optimal speed at each step using a no-regret algorithm (see for example [SB18]).

For any online policy π, the long run average *expected* energy consumption per time-unit for policy π under the probability transition P_s, noted Q_π, is defined as:

$$Q_\pi(x_0) = \mathbb{E}\left(\lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \mathit{Energy}(\pi(t))\right). \tag{8.64}$$

where x_0 is the initial state of the process, and *Energy*(s) is the energy consumption of the processor when the speed is s during one unit of time.

An optimal speed policy π^* minimizes the average expected energy consumption per time-unit given in Eq. (8.64). Therefore, the speed policy $\mathrm{(MP)}$ is defined as:

Definition 8.10 ($\mathrm{(MP)}$ policy)**.** *At each time $t \in \mathcal{T}$, the job that has the earliest deadline is executed at speed:*

$$\pi^{\mathrm{(MP)}}(t) \textit{ is such that } Q_{\pi^{\mathrm{(MP)}}}(x_0) = \inf_{\{\pi \mid \forall t\in\mathcal{T},\ \pi(t)\geq w_t^\pi(t+1)\}} Q_\pi(x_0). \tag{8.65}$$

Remark 8.4. *Several remarks are in order:*

- *The optimal policy minimizing the expected energy consumption may not be unique. In the following we consider one arbitrary such speed policy. This does not matter because feasibility as well as the expected energy consumption is the same for all of them.*
- *This definition of $\pi^{\mathrm{(MP)}}$ is not constructive but when the set of speeds is finite, then $\pi^{\mathrm{(MP)}}$ can be constructed explicitly using for example the Policy Iteration algorithm (see for instance [Put05]).*
- *It can also be shown that an optimal policy,* i.e. *a solution of Eq.* (8.65)*, is independent of x_0. This is outside the scope of this chapter.*

8.8.2 Feasibility analysis of $\mathrm{(MP)}$

Theorem 8.5 gives the value of $s_{\max}$ that ensures feasibility:

Theorem 8.5. $\mathrm{(MP)}$ *is feasible* $\iff s_{\max} \geq C$.

Proof. We distinguish the cases where the speed decision instants are integer and real numbers.

♦ The speed decision instants are integer numbers: $\mathcal{T} = \mathbb{N}$.

By definition, $\mathrm{(MP)}$ completes all the jobs before their deadline by construction: $\pi^{\mathrm{(MP)}}(t) \geq w_t^{\mathrm{(MP)}}(t+1)$. Therefore, $\mathrm{(MP)}$ is feasible if at any time $t \in \mathbb{N}$, $\pi^{\mathrm{(MP)}}(t) \leq s_{\max}$.

1. *Case $s_{\max} < C$:* In that case, no speed policy can guarantee feasibility as shown in Proposition 8.2.

2. *Case $s_{\max} \geq C$:* To prove the result, we first modify the *Energy* function as follows: For all speeds $s > s_{\max}$, we set *Energy*$(s) = \infty$. For $s \leq s_{\max}$, the *Energy* function remains unchanged. This modification is valid because the processor cannot use speeds larger than $s_{\max}$ anyway. Therefore, the energy consumption for such unattainable speeds can be arbitrarily set to any value. The benefit of using this modification is the following. Instead of constraining the speed to

remain smaller than $s_{\max}$, we let the scheduler use unbounded speeds, but this incurs an infinite consumption. A test to check if a policy uses speeds larger than $s_{\max}$ is that its average energy consumption will be infinite.

Starting from an empty system with no pending job, *i.e.* $x_0 = (0, 0, \ldots, 0)$, we define the following naive policy $\tilde{\pi}$:

$$\forall t \in \mathbb{N}, \tilde{\pi}(t) := c_t \quad \text{where } c_t = \sum_{J_i=(r_i,c_i,D_i)} \{c_i | r_i = t\}. \tag{8.66}$$

In other words, c_t is the amount of work that arrived at time t, which is by definition less than C. The policy $\tilde{\pi}$ is feasible because it never uses a speed larger than $C \leq s_{\max}$ and all work is executed as fast as possible (within one time slot after its arrival). Furthermore, since for any t, $\tilde{\pi}(t) \leq C$, its long run expected energy consumption per time unit satisfies $Q_{\tilde{\pi}}(x_0) \leq \mathit{Energy}(C)$.

The optimal policy, being optimal in energy, satisfies $Q_{\pi^{(\mathrm{MP})}}(x_0) \leq Q_{\tilde{\pi}}(x_0)$, hence $Q_{\pi^{(\mathrm{MP})}}(x_0) \leq \mathit{Energy}(C)$. Therefore, (MP) is feasible by construction and never uses a speed larger than $s_{\max}$.

♦ **The speed decision instants are real numbers: $\mathcal{T} = \mathbb{R}$.**

When the speed can be changed at any time $t \in \mathbb{R}$, the average expected energy consumption of a policy π becomes

$$Q_\pi(x_0) = \lim_{T\to\infty} \mathbb{E}\left(\frac{1}{T}\int_0^T \mathit{Energy}(\pi(t))dt\right). \tag{8.67}$$

When $s_{\max} < C$, then Proposition 8.2 says that no policy can be feasible, so neither is (MP).

Now, let us consider that $s_{\max} \geq C$. The optimal policy $\pi^{(\mathrm{MP})}$ is defined by taking the inf in Eq. (8.65), not over the set $\mathcal{A}^{\mathbb{N}} = \{\pi \,|\, \forall t \in \mathbb{N},\ \pi(t) \geq w_t^\pi(t+1)\}$ anymore, but over the set $\mathcal{A}^{\mathbb{R}} = \{\pi \,|\, \forall t \in \mathbb{R},\ w_t^\pi(t) \leq 0\}$. Since $\mathcal{A}^{\mathbb{N}} \subset \mathcal{A}^{\mathbb{R}}$ (see Prop. 8.3), it follows that $Q^{\mathcal{A}^{\mathbb{R}}}_{\pi^{(\mathrm{MP})}} \leq Q^{\mathcal{A}^{\mathbb{N}}}_{\pi^{(\mathrm{MP})}}$.

We have proven above that if $s_{\max} \geq C$, then $Q^{\mathcal{A}^{\mathbb{N}}}_{\pi^{(\mathrm{MP})}}$ is finite (it is less than $\mathit{Energy}(C)$). Therefore, $Q^{\mathcal{A}^{\mathbb{R}}}_{\pi^{(\mathrm{MP})}} \leq \mathit{Energy}(C)$. This implies that the optimal policy never uses speeds larger than $s_{\max}$, as in the discrete case. In conclusion, the (MP) policy with $\mathcal{T} = \mathbb{R}$ is feasible if and only if $s_{\max} \geq C$. □

8.9 Summary and Comparison of the four Policies

Table 8.1 summarizes the necessary and sufficient feasibility conditions on $s_{\max}$ for the four online speed policies (OA), (AVR), (BKP), and (MP), both in the integer and real speed decision instants cases.

For a given online speed policy π, we define the *feasibility region* $\mathcal{F}_\pi$ as the set of all triples $(C, \Delta, s_{\max})$ such that π is feasible. We rely on this notion of feasibility region to compare the policies. We make the following remarks:

Online speed policy	Necessary and sufficient feasibility condition	
speed decision instants	$\mathbb{N}$	$\mathbb{R}$
(OA)	$s_{\max} \geq C(h_{\Delta-1}+1)$	
(AVR)	$s_{\max} \geq Ch_{\Delta}$	
(BKP)	$s_{\max} \geq \frac{3}{2}(e-1)C$ ($\simeq 2.577\,C$)	$s_{\max} \geq eC$ ($\simeq 2.718\,C$)
(MP)	$s_{\max} \geq C$	

Table 8.1.: Necessary and sufficient feasibility condition of the four online speed policies.

1. By observing the (AVR) and (OA) feasibility bounds, we can remark that their maximal speeds are asymptotically identical when Δ becomes large. However, since for all $\Delta \in \mathbb{N}$ we have $\frac{1}{\Delta} \leq 1$, (AVR) and (OA) satisfy the following equation:

$$\pi^{(\mathrm{AVR})}_{\max} \leq \pi^{(\mathrm{OA})}_{\max}.$$

Consequently, since the maximal speed reached by (OA) is faster than the maximal speed reached by (AVR), (AVR) has a better feasibility than (OA), in the sense that the feasibility region of (AVR) includes the feasibility region of (OA):

$$\mathcal{F}_{(\mathrm{OA})} \subset \mathcal{F}_{(\mathrm{AVR})}.$$

2. Let us now compare the feasibility regions of (AVR) and (OA) with that of (BKP). Since this comparison depends on the harmonic number h_Δ, we display in Table 8.2 the approximated values of h_Δ and $h_{\Delta-1}+1$ (rounded down) for different values of Δ:

Δ	2	3	4	5	6	7	8	9
h_Δ	1.500	1.833	2.083	2.283	2.450	2.593	2.717	2.828
$h_{\Delta-1}+1$	2.000	2.500	2.833	3.083	3.283	3.450	3.593	3.717

Table 8.2.: Values of the harmonic numbers h_Δ and of $h_{\Delta-1}+1$ (with 3 significant digits).

Since the feasibility bounds of (BKP) are $\frac{3}{2}(e-1)C \simeq 2.577\,C$ when $\mathcal{T} = \mathbb{N}$ and $eC \simeq 2.718\,C$ when $\mathcal{T} = \mathbb{R}$, we compare in Table 8.3 the feasibility regions of (AVR), (OA), and (BKP) depending on the value of Δ:

Feasibility regions $\mathcal{F}_\pi$	$\mathcal{F}_{(\mathrm{AVR})} \subset \mathcal{F}_{(\mathrm{BKP})}$	$\mathcal{F}_{(\mathrm{OA})} \subset \mathcal{F}_{(\mathrm{BKP})}$	$\mathcal{F}_{(\mathrm{OA})} \subset \mathcal{F}_{(\mathrm{AVR})}$
Integer decision instants	$\forall \Delta \geq 7$	$\forall \Delta \geq 4$	$\forall \Delta \in \mathbb{N}$
Real decision instants	$\forall \Delta \geq 9$	$\forall \Delta \geq 4$	$\forall \Delta \in \mathbb{N}$

Table 8.3.: Feasibility region comparisons for (OA), (AVR), and (BKP).

(MP) is not present in Table 8.3 because it is clear from Table 8.1 that (MP) has the largest feasibility region:

$$\forall \pi \in \{(\mathrm{OA}), (\mathrm{AVR}), (\mathrm{BKP})\}, \quad \mathcal{F}_\pi \subset \mathcal{F}_{(\mathrm{MP})}$$

3. Unlike (OA) and (AVR), the (BKP) feasibility bounds are independent of the maximal deadline Δ. This means that the (BKP) feasibility regions do not change when Δ grows, whereas for (OA) and (AVR) the feasibility region decreases to the empty set when Δ increases.

4. For (BKP), one can wonder whether the parameter e can be changed in Eq. (8.48) to improve its feasibility (see Theorems 8.3 and 8.4). If we replace e by a parameter α in the definition of (BKP), we obtain a variant policy denoted (BKP_α). The feasibility region becomes $\mathcal{F}_{(\mathrm{BKP}_\alpha)} = \{s_{\max} \geq \alpha C\}$ for any $\alpha \geq e$, by using the same proof as in Section 8.7. However, if $\alpha < e$, then it can be shown that (BKP_α) is not feasible even with $s_{\max} = +\infty$. It follows that $\alpha = e$ is the best possible choice.

5. Finally, (MP) is optimal both in terms of energy and feasibility, so it is a good candidate to be used online to process real-time jobs. Its drawback, however is twofold: on the one hand its complexity, the time and space complexity to compute $\pi^{(\mathrm{MP})}(t)$ being $O(C^{\Delta})$ (see [GGP17]); and on the other hand the requirement to know the probability distributions on r_i, c_i, and d_i.

8.10 Conclusion

Adjusting the processor speed dynamically in hard real-time systems allows the energy consumption to be minimized. This is achieved by an *online speed policy*, the goal of which is to determine the speed of the processor to execute the current, not yet finished, jobs. Several such policies have been proposed in the literature, including (OA), (AVR), (BKP), and (MP). Since they are targeting hard real-time systems, they must satisfy two constraints: each real-time job must finish before its deadline, and the maximal speed used by the policy must be less than or equal to the maximal speed $s_{\max}$ available on the processor. We call the conjunction of these two constraints the *feasibility* condition of the policy.

In this chapter, we have established for each of the four policies (OA), (AVR), (BKP), and (MP), a necessary and sufficient condition for the feasibility. (OA) is feasible if and only if $s_{\max} \geq C(h_{\Delta-1}+1)$. (AVR) is feasible if and only if $s_{\max} \geq Ch_{\Delta}$. (BKP) is feasible if and only if $s_{\max} \geq eC$ when the processor speed can change at any time, and $s_{\max} \geq \frac{3}{2}(e-1)C$ when the processor speed can change only upon the arrival of a new job (for the other policies, the times at which the processor speed can change has no impact on the feasibility condition). Finally, (MP) is feasible if and only if $s_{\max} \geq C$. This is optimal because, as shown in Proposition 8.2, the necessary condition of feasibility of all online policies is $s_{\max} \geq C$. Therefore, (MP) is optimal in terms of feasibility in addition to being optimal in energy (on average), but it requires the statistical knowledge of the arrival times, execution times, and deadlines of the jobs, and it is more expensive to compute than the other speed policies.

Appendix

8.11 Appendix A: Uniqueness of the solution of Eq. (8.37)

Let us rewrite Eq. (8.37) $\forall t \in [k, k+1[$ as follow:

$$\pi(t) = \max_{v>t} \frac{w_k(v) - \int_k^t \pi(x)\mathrm{d}x}{v-t}. \tag{8.68}$$

The goal of this part is to prove that there exists a unique solution π for this equation. By doing an appropriate variable shift on Eq. (8.68), $u = v - t$, we obtain:

$$\pi(t) = \sup_{u>0} \frac{w_k(u+t) - \int_k^t \pi(x)\mathrm{d}x}{u}. \tag{8.69}$$

By defining $W(t) = \int_k^t \pi(x)\mathrm{d}x$, and integrating Eq. (8.69) between k and t, we obtain:

$$W(t) = \int_k^t \sup_{u>0} \frac{w_k(u+s) - W(s)}{u} \mathrm{d}s. \tag{8.70}$$

Let us now define the function $F(s,x)$ as follows:

$$F(s,x) = \sup_{u>0} \left(\frac{f_s(u) - x}{u} \right) \tag{8.71}$$

where $f_s(.) = w_k(. + s)$. Then $f_s(.)$ is such that:

1. f_s is an increasing function bounded by $C\Delta \in \mathbb{R}^+$.
2. $f_s(0) = w_k(s) = 0$ because $w_k(k) = 0$ by feasibility and no job arrives between k and s.
3. f_s satisfies:

$$\lim_{\substack{t\to 0\\ t\geq 0}} \frac{f_s(t)}{t} = 0 \tag{8.72}$$

   because $w_k(s)$ is constant for $s \in [k, k+1)$.

The function $W(t) = \int_k^t \pi(u)\mathrm{d}u$ also satisfies the following integro-differential equation:

$$W(t) = \int_k^t F(s, W(s))\mathrm{d}s. \tag{8.73}$$

Lemma 8.2. *There exists a unique solution W to Eq.* (8.73).

Proof. First, let us show in Lemmas (8.3)-(8.4) that the function $F(s,x)$ is Lipschitz in x.

Lemma 8.3. *Let $t_0 > 0$ be the first time such that the sup of $F(s,0)$ is reached. Then $F(s,x)$ is a $\frac{1}{t_0}$-Lipschitz function in x.*

Proof. For the proof of Lemma 8.3, we will note $g(x) = F(s, x)$. Let $x, y \in [0, a]$ (where a is an arbitrary positive number). We want to prove that:

$$\exists k \in \mathbb{R},\ |g(x) - g(y)| \leq k|x - y|. \tag{8.74}$$

Let us compute the difference $|g(x) - g(y)|$:

$$|g(x) - g(y)| \quad = \quad \left| \sup_{t>0} \frac{f_s(t) - x}{t} - \sup_{t>0} \frac{f_s(t) - y}{t} \right| \tag{8.75}$$

Since, by assumption, the function $f_s(t)$ is bounded by a, the sup for $g(x)$ is reached for a certain value of t, noted t_x.

By definition of the $\sup$, we have $\sup_{t>0} \frac{f_s(t)-y}{t} \geq \frac{f_s(t_x)-y}{t_x}$, hence Eq. (8.75) becomes:

$$\begin{aligned} |g(x) - g(y)| &\leq \left| \frac{f_s(t_x) - x}{t_x} - \frac{f_s(t_x) - y}{t_x} \right| \\ |g(x) - g(y)| &\leq \left| \frac{y - x}{t_x} \right| \end{aligned} \tag{8.76}$$

Now let us prove Lemma (8.4), which states that t_x is an increasing function in x.

Lemma 8.4. *Let t_x be the function of x such that:*

$$\forall a \in \mathbb{R},\ \forall x \in [0, a],\ t_x : x \longmapsto \operatorname*{argmax}_t \frac{f_s(t) - x}{t}.$$

Then t_x is an increasing function of x.

Proof. Let $x, y \in [0, a]$ such that $x \leq y$, and let t_y be such that:

$$\frac{f_s(t_y) - y}{t_y} = \max_t \left(\frac{f_s(t) - y}{t} \right) \tag{8.77}$$

The goal is to prove Eq. (8.78) below:

$$\frac{f_s(t) - x}{t} \leq \frac{f_s(t_y) - x}{t_y}. \tag{8.78}$$

By definition of the $\max$, we have for any defined function f_s:

$$\forall t \in \mathbb{R},\ f_s(t) \leq \frac{f_s(t_y) - y}{t_y} t + y. \tag{8.79}$$

We now define two lines:

- the line L_1 that corresponds to the slope for the maximal value of y, *i.e.* the line that links the points $(0, y)$ and $(t_y, f_s(t_y))$; its equation corresponds to the left part of Eq. (8.79);

- and the line L_2 that links $(0, x)$ on the ordinate axis and the point $(t, f_s(t))$.

The functions $t \longmapsto L_1(t)$ and $t \longmapsto L_2(t)$ correspond to all the points of their respective lines.

By definition of $L_1(t)$, we have $f_s(t) \leq L_1(t)$. Moreover as time $t \geq t_y$, by construction of line L_2, we have $L_2(t) \leq L_1(t)$. Since $x \leq y$, we also have $L_2(0) \leq L_1(0)$. All these inequalities on some points of the two lines L_1 and L_2 imply that $L_1(t_y) \geq L_2(t_y)$. The expressions of the functions $L_1(t)$ and $L_2(t)$ lead to the following inequality:

$$f_s(t_y) \geq \frac{f_s(t) - x}{t} t_y + x \iff \frac{f_s(t_y) - x}{t_y} \geq \frac{f_s(t) - x}{t}.$$

Eq. (8.78) is therefore satisfied, and so the function $x \longmapsto t_x$ is an increasing function.

□

Using Eq. (8.72), the fact that f_s is an increasing function, and the fact that $f_s(0) = 0$, the first time t such that $F(t, 0) > 0$ is strictly larger than 0, and as we want to determine for all t the sup of $F(t, 0)$, then t_0 is strictly positive.

Since t_x is an increasing function of x by Lemma (8.4), and since $t_0 > 0$, then Eq. (8.76) becomes:

$$|g(x) - g(y)| \quad \leq \quad \frac{1}{t_0}|y - x|. \tag{8.80}$$

Eq. (8.80) concludes that g is $\frac{1}{t_0}$-Lipschitz.

□

Since $F(s, x)$ is Lipschitz in x, the Picard-Lindelof theorem allows us to concludes that there exists a unique solution $W(t)$ for the Eq. (8.73).

Therefore Eq. (8.73) can be rewritten as follow:

$$\pi(t) \quad = \quad \sup_{u>0} \frac{f_s(u) - W(t)}{u}. \tag{8.81}$$

By Eq. (8.81), π is a function of W, so π is also unique. □

8.12 Appendix B: Concavity of the executed work by (OA) for a given w

In this appendix we provide a more exhaustive study of the speed policy (OA). We show that the work executed by (OA) is the convex envelope of the graph of the remaining work function $w(.)$, when $w(.)$ is fixed (*i.e.* all the jobs arrive at time 0). Using the same notation as in the previous appendix (the index $k = 0$ is dropped in w_0), we define $W(t) = \int_0^t \pi^{(\mathrm{OA})}(u)\mathrm{d}u$, the amount of work executed by (OA) from time 0 to time t. It is such that:

$$W(t) = \int_0^t \sup_{u \geq 0} \frac{w(u+s) - W(s)}{u} \mathrm{d}s = \int_0^t \sup_{v \geq s} \frac{w(v) - W(s)}{v - s} \mathrm{d}s. \tag{8.82}$$

$W(t)$ corresponds to the quantity of work executed between 0 and t, and the goal of this part is to show that $W(t)$ is the smallest concave function that is above $w(t)$.

Lemma 8.5. *Let w be any real non-decreasing function that admits right-derivatives everywhere (not necessarily staircase), with $w(0) = 0$. Then $W(t)$ as defined in Eq.* (8.82) *satisfies the following properties:*

1. *W is continuous, $W(0) = 0$, and $\forall t \geq 0$, $W(t) \geq w(t)$.*
2. *W is non-decreasing in w.*
3. *If w is concave, then $W(t) = w(t)$.*
4. *W is concave.*
5. *$W = \widehat{w}$ where $\widehat{w}$ is the convex hull of w.*

Proof.

1. $W(t)$ being an integral from 0 to t, W is continuous, $W(0) = 0$, and W has right-derivatives everywhere: $W'_+(t) = \sup_{u \geq 0} \frac{w(u+t) - W(t)}{u}$. Let us denote by $w'_+(.)$ the right-derivative of w: $w'_+(t) = \lim_{u \to 0, u \geq 0}(w(t+u) - w(t))/u$. Then $w'_+(t) \leq \sup_{u \geq 0} \frac{w(u+t) - w(t)}{u}$. Since $w(0) = 0 = W(0)$, then by Petrovitsch Theorem on differential inequalities, [Pet01], we have $W(t) \geq w(t)$ for all $t \geq 0$.

2. By definition of the function W, it is a non-decreasing function in w.

3. Let us suppose that w is concave. By replacing, in the right part of Eq. (8.82), W by w, one gets inside the integral:

$$\sup_{v \geq s} \frac{w(v) - w(s)}{v - s}. \tag{8.83}$$

Since w is concave by assumption, it is right and left differentiable at any point t. This means that w'_+, the right-derivative of w, is decreasing. Therefore the sup is reached when v goes to s. We thus have:

$$\sup_{v \geq s} \frac{w(v) - w(s)}{v - s} = w'_+(s). \tag{8.84}$$

By using Eq. (8.84) and replacing in Eq. (8.82), we obtain:

$$\int_0^t \sup_{v \geq s} \frac{w(v) - w(s)}{v - s} \mathrm{d}s = \int_0^t w_+^{'}(u)\mathrm{d}u = w(t). \tag{8.85}$$

The last equality in Eq. (8.85) is due to the fact that w is concave. Indeed, since w is concave, its derivative is defined on the whole interval $[k, t]$ except for a finite number of values u. The integral does not depend on these points, so we have Eq (8.85).

To conclude, w is a solution of Eq (8.82) and so $W = w$ by uniqueness of the solution.

4. For any $t \geq 0$, let L_t be the right tangent of W at the point t. The equation of the line L_t is: $L_t(v) = W(t) + W'_+(t)(v - t)$. Since $W(t) \geq w(t)$, we have for all $v \geq t$, $L_t(v) = W(t) + W'_+(t)(v - t) \geq w(v)$.

   If we replace w by L_t in the definition of W, we get a new function W_L that is larger than W by item 2 and is equal to L_t by item 3. This means that W is below its right tangents.

   Now, this implies that W is concave: By contradiction, let $x < y$ be such that $\forall z \in [x, y]$, we have $W(z) < A(z)$, where $A(.)$ is the affine interpolation between $W(x)$ and $W(z)$. Since W is below its right tangents, then $W(y) \leq W(z) + W'_+(z)(y - z)$. This implies that $W'_+(z)$ is larger than the slope of $A(.)$: in other words, $W'_+(z) \geq \frac{W(y)-W(x)}{y-x}$. By integrating this inequality from x to z, we get $W(z) \geq A(z)$. This contradicts the initial assumption that $W(z) < A(z)$.

   The final conclusion is that W is always above its affine interpolation, hence W is concave.

5. Let us prove first that $W \geq \widehat{w}$. By definition of the convex hull, we know that $w \leq \widehat{w}$, and as W is an increasing function in w, $W \leq W_{\widehat{w}}$, where $W_{\widehat{w}}$ is the function W where w is replaced by $\widehat{w}$. Since $\widehat{w}$ is concave, we then have by item 3 the fact that $\widehat{w} = W_{\widehat{w}}$. This implies:

   $$W \leq \widehat{w}.$$

   Now we prove the other inequality, *i.e.* that $\widehat{w} \leq W$. By item 1, we get $W \geq w$. By item 4, we know that W is concave. Since $\widehat{w}$ is the smallest concave function above w, we finally have:

   $$W \geq \widehat{w}.$$

   We therefore conclude that: $W = \widehat{w}$.

□

9 Conclusion

9.1 General Conclusion

In this thesis several algorithms that optimize the energy consumption of a single unicore processor that executes real-time jobs have been presented. The offline and online cases have been studied. For the offline case, we present an algorithm that solves the same problem: determining the best speed schedule to execute n jobs before their deadline while minimizing the total energy consumption. We show that it can be done in constant time. Furthermore the theoretical lower bound is also linear in the maximal deadline and the size of the jobs. The online case is separated into several situations, that depend on the knowledge we have of the job features. It depends more particularly on the information we have on active and future jobs. The online case has already been studied in the literature, however taking advantage of the potential information we could have on the future jobs has attracted much attention in this community. In this thesis, three cases have been studied, each of them considers that past jobs are known: the clairvoyant active jobs with statistical information for future jobs case is analyzed in Chapter 4, then the non-clairvoyant active jobs with statistical information for future jobs case is seen in Chapter 5. And to finish Chapter 6 and 7 have presented the clairvoyant active jobs without any information on future jobs case. Statistical information about future jobs may be collected by using past experiments or simulations, or by analyzing the structure of the job features. We have shown in this document, for each cases, how to compute the optimal speed schedule. This leads to a gain in energy in comparison with the solutions that do not take advantage of these information. In each case, our solutions provide performances that are close to the optimal offline solutions on average, and outperforms classical online solutions. This is also true for the learning cases, but it depends on the length of the learning period we consider.

In the last chapter we analyzed the feasibility of the online policies. It highlights the fact that the policies we propose are better in feasibility than the existing online policies from the literature.

9.2 Future Work

There are four main points in the near future that can be explored:

- To fight against the state space explosion. During all this thesis, we have shown that the state space size is critical. One way we propose to solve this problem is to group some states and apply our algorithms on this reduced state space. One extension will be to implement this method and characterize the "good" way to reduce the state space. In previous chapters, we gave some ideas on how to do a coarse discretization, and one future research direction will be to implement it.

- To run the presented algorithms on realistic benchmarks and not on synthetics one.
- To use more accurate learning algorithm to learn the transition probability matrix. We can compare these algorithms that exist in the literature with results from Chapter 6, by using Bayesian methods to estimate the transition probability matrix. This method will be more interesting, because it does not need an absolute training period and improve the choices during the algorithm evolution.
- To search how to accelerate the convergence of the Q-learning algorithm. Some algorithms exist in the literature, but they either have the same time complexity, or they are applicable only to very small state spaces.

More broadly, this thesis builds a bridge between the real-time systems and the optimization communities, by using Markov Decision Processes to decrease the energy consumption. Matching real-time systems and optimization techniques with MDP methods could be used in the future to solve other real-time problems.

Appendix A

A.1 Optimality of EDF Scheduling

In this appendix, we generalize a known result about the optimality of EDF schedulability [Hor74] to the case where the processor speed varies. If EDF policy is feasible, therefore all other optimal policies are feasible. Here we have to show this optimality under the condition of a processor speed variability.

Proposition A.1. *For any processor speed profile, the scheduling policy Earliest Deadline First (EDF) minimizes the maximum lateness of any set of jobs as described in section 4.2.1 among all scheduling policies.*

Optimal schedulability is equivalent to having a maximum of lateness equal to 0. The proof of proposition A.1 is done in [Hor74] when the speed is constant ($s = 1$), which is not the case here. Indeed, speed varies over time, is independent of the active job. We use the idea of Horn's proof to demonstrate this property.

Proof.
Recall that jobs are defined as $J_i = (r_i, c_i, d_i)$. We denote also $s(t)$ the speed function, which is assumed to be integrable and finite (this is the only assumption on speeds). For any scheduler ρ, we introduce the following definitions:

- The job executed at time t is $\rho(t)$.
- The finishing time of job J_i under ρ, is noted $f_\rho(J_i)$.
- The remaining work, $w_\rho(J_i, t)$, is the quantity of work of job J_i that remains to be done at time t under ρ:
$$w_\rho(J_i, t) = c_i - \int_0^t s(u)\mathbb{1}_{\{\rho(u)=J_i\}}\mathrm{d}u$$
where $\mathbb{1}_{\{\rho(u)=J_i\}} = 1$ if $\rho(u) = J_i$ and 0 otherwise.
- The lateness of job J_i under ρ is:
$$\lambda_\rho(J_i) = (f_\rho(J_i) - d_i)^+$$
- The maximum of lateness of schedule ρ is:
$$\Lambda(\rho) = \max_{J_i} \lambda_\rho(J_i)$$

From now on, we consider two schedulers, EDF and an arbitrary scheduler ρ. For ρ, let us consider the time instants when the scheduler switches from one job execution to another. For EDF, let us also consider the time instants when EDF switches from one job to another. By combining these two sets of time instants, we denote by $t_1 < \cdots < t_m$, all the time instants when either EDF or ρ switches its job.

Let us now focus on the first time t_k when ρ and EDF differ: we denote by J the job (represented in red in Figure A.1) executed by ρ and by J' the job (represented in blue in Figure A.1) executed by EDF. The fact that EDF chooses J' over J at time t_k implies that the respective deadlines of J and J' are such that

$$d_J \geq d_{J'} \tag{A.1}$$

We denote by W_k the work quantity executed between t_k and t_{k+1}, *i.e.*, $W_k = \int_{t_k}^{t_{k+1}} s(t)\mathrm{d}t$. Since both jobs J and J' are executed between t_k and t_{k+1}, and since EDF and ρ coincide up to time t_k, the remaining work for both of them at time t_k must be larger than W_k:

$$w_\rho(J, t_k) = w_{\text{EDF}}(J, t_k) \geq W_k$$
$$w_\rho(J', t_k) = w_{\text{EDF}}(J', t_k) \geq W_k$$

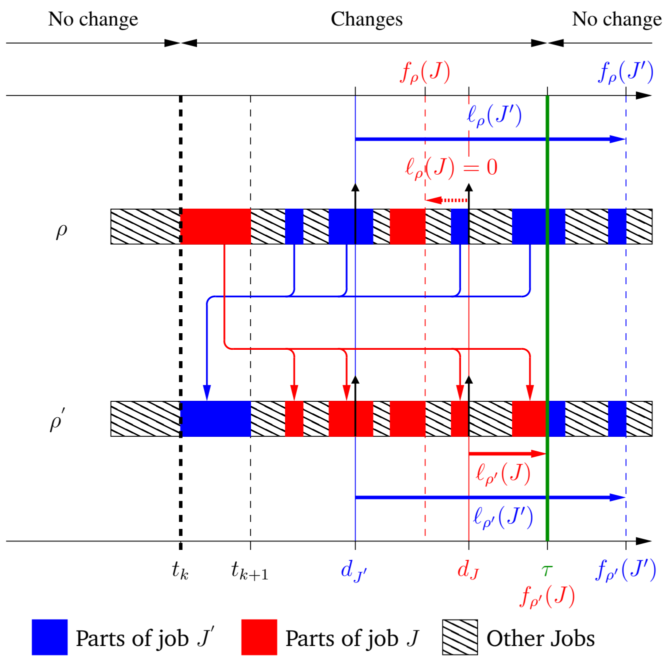


Figure A.1.: Schedules ρ and ρ' and their respective lateness on $[t_k, \tau]$. The lateness of J' is the same under ρ and ρ' while the lateness of J increases under ρ', but remains smaller than the lateness of J' under ρ. So the maximum of lateness is the same on ρ and ρ'.

Let us define a new scheduler ρ' as follows:

- $\forall t \in [0, t_k]$, $\rho'(t) = \rho(t) = EDF(t)$.

- $\forall t \in [t_k, t_{k+1}]$, $\rho'(t) = J'$. Since the function $g : t, J \longrightarrow \int_{t_k}^{t} s(u)\mathbb{1}_{\{\rho(u)=J\}}\mathrm{d}u$ is a continuous function of t and because $w_\rho(J', t_{k+1}) \geq W_k$, by the Intermediate Value Theorem, there exists τ such that:
$$\tau = \inf\left\{t \,\middle|\, \int_{t_{k+1}}^{t} s(u)\mathbb{1}_{\{\rho(u)=J'\}}\mathrm{d}u = W_k\right\} \tag{A.2}$$

- Between times t_{k+1} and τ, the scheduler ρ' executes J whenever ρ executes J': $\forall t_{k+1} \leq t \leq \tau$, $\rho(t) = J' \Rightarrow \rho'(t) = J$ and $\rho(t) \neq J' \Rightarrow \rho'(t) = \rho(t)$.

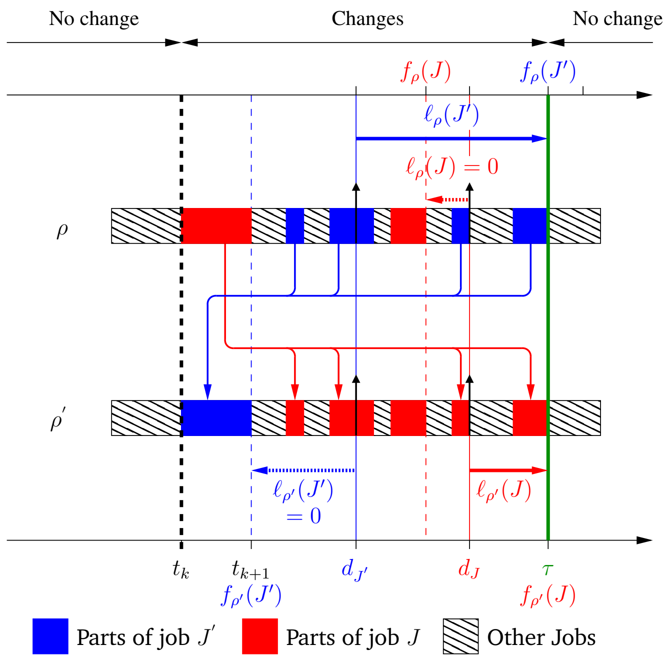


Figure A.2.: The lateness of $J^{'}$ is better under ρ' than under ρ while the lateness of J increases under ρ', but remains smaller than the lateness of J' under ρ. So the maximum of lateness is smaller under ρ' than under ρ.

Now let us show that the maximum lateness of ρ' is smaller or equal than the maximum lateness of ρ. First, the latenesses of all jobs except J and J' do not change under ρ'.

Now, let us analyze the lateness of job J' under ρ'. By construction of ρ', the remaining work W_k is such that $w_\rho(J', t_k) \geq W_k$. So we are faced with two cases:

- The first one is $w_\rho(J', t_k) > W_k$. Here, all the remaining work due to job J' is not finished at $[t_k, \tau]$, then the finishing time is the same under both schedulers, so $\lambda_{\rho'}(J') = \lambda_\rho(J')$. This case is the case represented in Figure A.1.

- The second one is the case where $w_\rho(J', t_k) = W_k$. In this case, in t_{k+1}, the job J' has been entirely executed, so we have:

$$f_{\rho'}(J') = t_{k+1} \leq f_\rho(J') \leq f_\rho(J')$$

The two previous cases imply that we have no lateness increase for job J', so:

$$\lambda_{\rho'}(J') \leq \lambda_\rho(J') \leq \Lambda(\rho). \tag{A.3}$$

This case is the case represented in Figure A.2.

Lastly, let us analyze job J. Again there are two cases for the lateness $\lambda_{\rho'}(J)$ of job J under ρ':

- $f_\rho(J) > \tau$. This means that the execution of job J ends after the time τ. In that case, there is no difference for the finishing time of J whatever the scheduling. Indeed the difference between ρ and ρ' only modifies the order of the execution of parts of job J, which belongs to $[t_k, \tau]$, but not after time τ (see Figure A.1), so:

  $$f_{\rho'}(J) = f_\rho(J)$$

  The lateness is also the same:

  $$\lambda_{\rho'}(J) = \lambda_\rho(J) \leq \Lambda(\rho) \tag{A.4}$$

- $f_\rho(J) \leq \tau$. In that case, the end of the job J under ρ' occurs exactly at τ (see Figure A.1):

  $$f_{\rho'}(J) = \tau$$

  So the lateness of job J under ρ' is:

  $$\lambda_{\rho'}(J) = (\tau - d_J)^+$$

  On the other hand, we know that at time τ, ρ is executing job J', therefore, $f_\rho(J') \geq \tau$ and we also know that $d_{J'} \leq d_J$ (see (A.1)). We can conclude that:

  $$\lambda_{\rho'}(J) \leq (f_\rho(J') - d_{J'})^+ = \lambda_\rho(J') \leq \Lambda(\rho) \tag{A.5}$$

As a consequence, the maximum lateness does not increase under ρ' in comparison with ρ: By Eqs. (A.3), (A.4), and (A.5), we have:

$$\Lambda(\rho') \leq \Lambda(\rho)$$

If we repeat this reasoning starting with ρ' instead of ρ, then the new schedule will coincide with EDF further in time and the maximum lateness will not increase.

This shows that, eventually, EDF too minimizes the maximum lateness:

$$\Lambda(EDF) \leq \Lambda(\rho') \leq \Lambda(\rho)$$

which concludes the proof. □

A.2 Size of the State Space

This appendix is dedicated to the enumeration of the total number of states of the MDP.

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: $x_1 = w(\Delta) - w(\Delta - 1)$, and more generally, $x_j = w(\Delta - j + 1) - w(\Delta - j)$, for all $j = 1, \ldots, \Delta$, the released work being of maximal size C at any time, $x_1 \leq C$ because x_1 must be bounded by the amount of work that was released at step t. Similarly, $x_1 + x_2$ must be bounded by the amount of work that was released at steps t and $t - 1$, namely $2C$, and so on and so forth up to $x_1 + x_2 + \cdots + x_\Delta \leq \Delta C$. 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 $(x_1, x_2, ..., x_\Delta)$ satisfy the following conditions:

$$\begin{cases} x_1 \leqslant C \\ x_1 + x_2 \leqslant 2C \\ x_1 + x_2 + x_3 \leqslant 3C \\ \vdots \\ x_1 + x_2 + ... + x_\Delta \leqslant \Delta C \end{cases}$$

By defining the partial sums $y_j = x_1 + \cdots + x_j$, the number of states satisfies:

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

This multiple sum can be seen as a generalized Catalan number. Indeed, notice that a state characterized by its steps $(x_1, \ldots, x_\Delta)$ 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. A.3).

Counting the number of such paths has been done in [HP91] and corresponds to the generalized Catalan numbers, $C_n^k = \frac{1}{nk+1}\binom{nk+1}{n}$.

We propose below a direct proof for computing $Q(C, \Delta)$, which is new up to our knowledge. This new proof is inspired by the ordinary Catalan numbers. First, let us count the total number of paths from $(0, 0)$ to $(\Delta + 1, C(\Delta + 1))$ without the constraint of staying below the diagonal. This is standard combinatorics and this number of paths is equal to $\binom{(C+1)(\Delta+1)}{\Delta+1}$.

Second, this set of paths can be partitioned into *classes* according to the number of vertical steps taken *above* the diagonal. According to this classification, $Q(C, \Delta)$ is the size of class 0 (the paths that take no step over the diagonal).

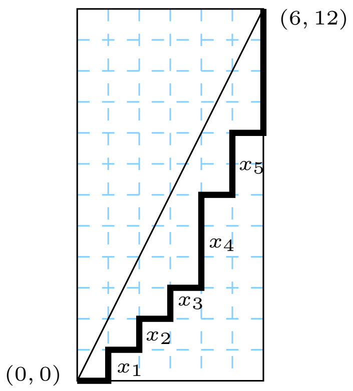


Figure A.3.: A valid state seen as a path below the diagonal $(0,0)$—$(6,12)$, for $C = 2$ and $\Delta = 5$.

Third, by using a *shift*, we will show that any path of class k (with $k < C$) can be bijectively transformed into a path of class $k+1$. This will prove that all the classes have the same size. Here is how to proceed.

Since the path is not of class C, it takes excursions below the diagonal and therefore, its first vertical step that hits the diagonal from below is well defined. The path P can be written xvy where v is this vertical step, and x and y are respectively the prefix and the suffix of P w.r.t. v. This is illustrated in Fig. A.4 (left).

We then construct the path P' by swapping the prefix and the suffix of P, *i.e.*, $P' = yvx$, and we claim that P' is in class $k+1$. The construction of P' from P is illustrated in Fig. A.4.

- The number of vertical steps in y is the same in P' and in P because y starts on the diagonal and ends on the diagonal in both cases (see Fig. A.4).
- Regarding x, it is *shifted up* by one step up in P', so all the vertical steps taken by x above the diagonal in P are still taken above the diagonal in P'. As for the vertical steps taken below the diagonal by x in P, they remain below the diagonal in P'. Indeed, suppose that there exists a vertical step in x that is below the diagonal in P but above the diagonal in P'. Since x is shifted up by one step, this means that this step was touching the diagonal from below in P. This is not possible since the first such step is v, hence not in x.

As a result, like y, x also contributes the same number of vertical steps above the diagonal in P and in P'. It follows that the only difference in the number of vertical steps between P and P' comes from v, which is not above the diagonal in P but is above in P'. Hence the class of P' is $k+1$.

Let us now show that this transformation is bijective. As explained above, the last sub-path x in P' does not contain any vertical step that starts on the diagonal. This means that the step v is the last vertical step that starts from the diagonal in P'. This implies that P can be reconstructed back from P'. Therefore, the transformation of P into P' is reversible, so it is an injection. Since all paths of class $k+1$ contain a last vertical step starting on the diagonal, the transformation is bijective.

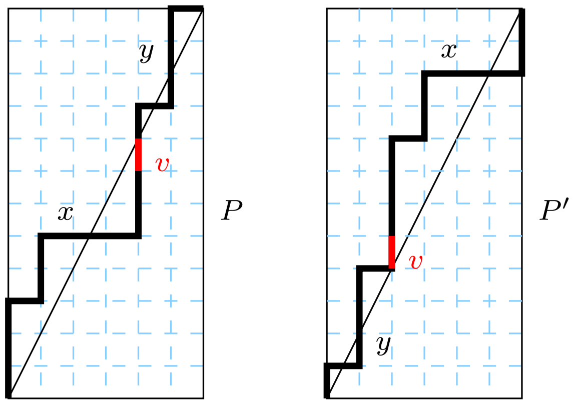


Figure A.4.: The left figure shows a path P (with $\Delta{+}1{=}6$ and $C{=}2$) which belongs to class 8 (*i.e.*, it takes 8 steps above the diagonal). The red step (denoted v) is the first vertical step that hits the diagonal from below. By swapping the prefix x and the suffix y of P, the class of the resulting path becomes 9. The resulting path $P'{=}yvx$ is displayed on the right.

The construction of this bijection means that the size of class k is equal to the size of class $k+1$, for all $0 \leq k < C$. This means that all the classes have the same size. Therefore, the class 0 has size $Q(C,\Delta) = \frac{1}{1+C(\Delta+1)}\binom{(C+1)(\Delta+1)}{\Delta+1}$.

Using the Stirling formula, we finally get $Q(C,\Delta) \approx \frac{e}{\sqrt{2\pi}}\frac{1}{(\Delta+1)^{3/2}}(e\,C)^{\Delta}$.

Bibliography

[95] *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*. IEEE Computer Society, 1995.

[ABH16] Karim Abbas, Joos Berkhout, and Bernd Heidergott. *A Critical Account of Perturbation Analysis of Markov Chains*. Tech. rep. arXiv:1609.04138v1. arXiv, 2016.

[AIS04] J. Augustine, S. Irani, and C. Swamy. „Optimal Power-Down Strategies“. In: *Symposium on Foundations of Computer Science, FOCS'04*. Rome, Italy: IEEE, Oct. 2004, pp. 530–539. cit. on p. 147

[AO06] Peter Auer and Ronald Ortner. „Logarithmic Online Regret Bounds for Undiscounted Reinforcement Learning“. In: *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*. Ed. by Bernhard Schölkopf, John C. Platt, and Thomas Hofmann. MIT Press, 2006, pp. 49–56. cit. on p. 111

[AOM17] Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. „Minimax Regret Bounds for Reinforcement Learning“. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 263–272.

[Ayd+01] Hakan Aydin, Rami G. Melhem, Daniel Mossé, and Pedro Mejıéa-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: IEEE Computer Society, June 2001, pp. 225–232. cit. on p. 42

[Aza+11] Mohammad Gheshlaghi Azar, Rémi Munos, Mohammad Ghavamzadeh, and Hilbert J. Kappen. „Speedy Q-Learning“. In: *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.* 2011, pp. 2411–2419. cit. on p. 130

[BB00] T. Burd and R. Brodersen. „Design Issues for Dynamic Voltage Scaling“. In: *International Symposium on Low Power Electronics and Design, ISLPED'00*. Rapallo, Italy, July 2000. cit. on pp. 35, 38, 66

[BBA10] A. Bastoni, B. Brandenburg, and J.H. Anderson. „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*. July 2010, pp. 33–44. cit. on p. 67

[BG16] B. Brandenburg and M. Gul. „Global Scheduling Not Required: Simple, Near-Optimal Multiprocessor Real-Time Scheduling with Semi-Partitioned Reservations“. In: *Real-Time Systems Symposium, RTSS'16*. IEEE Computer Society, Dec. 2016, pp. 99–110. cit. on p. 67

[BKP07] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. „Speed Scaling to Manage Energy and Temperature“. In: *Journal of the ACM* 54.1 (2007). cit. on pp. 4, 41, 42, 44, 81, 93, 146, 165, 167

[BM00] Vivek S. Borkar and Sean P. Meyn. „The O.D.E. Method for Convergence of Stochastic Approximation and Reinforcement Learning“. In: *SIAM J. Control and Optimization* 38.2 (2000), pp. 447–469. cit. on p. 129

[BM98] François Baccelli and Jean Mairesse. „Ergodic Theorems for Stochastic Operators and Discrete Event Networks“. In: *Idempotency*. Ed. by Cambridge University Press. Publications of the Newton Institute. Cambridge University Press, 1998, pp. 171–208.

[Bra11] B. Brandenburg. „Scheduling and Locking in Multiprocessor Real-Time Operating Systems“. PhD thesis. Chapel Hill (NC), USA: The University of North Carolina at Chapel Hill, 2011. cit. on p. 67

[BS09] Enrico Bini and Claudio Scordino. „Optimal two-level speed assignment for real-time systems“. In: *IJES* 4.2 (2009), pp. 101–111. cit. on pp. 42, 82, 92

[BSA14] M. Bandari, R. Simon, and H. Aydin. „Energy Management of Embedded Wireless Systems Through Voltage and Modulation Scaling Under Probabilistic Workloads“. In: *International Green Computing Conference, IGCC'14*. Dallas, TX, USA: IEEE Computer Society, Nov. 2014, pp. 1–10. cit. on p. 35

[BT96] Dimitri Bertsekas and John Tsitsiklis. *Neuro-dynamic programming*. Belmont, Mass., USA: Athena Scientific, 1996. cit. on p. 130

[Bur+13] Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, eds. *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. 2013.

[CST09] J.-J. Chen, N. Stoimenov, and L. Thiele. „Feasibility Analysis of On-Line DVS Algorithms for Scheduling Arbitrary Event Streams“. In: *Real-Time Systems Symposium, RTSS'09*. Washington (DC), USA: IEEE, Dec. 2009, pp. 261–270. cit. on p. 147

[DM17] Adithya M. Devraj and Sean P. Meyn. „Zap Q-Learning“. In: *Annual Conference on Neural Information Processing Systems*. Long Beach (CA), USA, Dec. 2017, pp. 2232–2241. cit. on p. 130

[GGP17] Bruno Gaujal, Alain Girault, and Stephan Plassart. *Dynamic Speed Scaling Minimizing Expected Energy Consumption for Real-Time Tasks*. Tech. rep. hal-01615835. Inria, 2017. cit. on pp. 17, 82, 83, 84, 146, 171, 175

[GGP19a] Bruno Gaujal, Alain Girault, and Stéphan Plassart. *A Discrete Time Markov Decision Process for Energy Minimization Under Deadline Constraints*. Research Report RR-9309. Grenoble Alpes ; Inria Grenoble Rhône-Alpes, Université de Grenoble, Dec. 2019. cit. on p. 41

[GGP19b] Bruno Gaujal, Alain Girault, and Stéphan Plassart. „A Linear Time Algorithm for Computing Off-line Speed Schedules Minimizing Energy Consumption“. In: *MSR 2019 - 12ème Colloque sur la Modélisation des Systèmes Réactifs*. Angers, France, Nov. 2019, pp. 1–14. cit. on p. 17

[GGP19c] Bruno Gaujal, Alain Girault, and Stéphan Plassart. *Exploiting Job Variability to Minimize Energy Consumption under Real-Time Constraints*. Research Report RR-9300. Inria Grenoble Rhône-Alpes, Université de Grenoble ; Université Grenoble - Alpes, Nov. 2019, 23p. cit. on p. 81

[GGP20a] Bruno Gaujal, Alain Girault, and Stéphan Plassart. *A Linear Time Algorithm Computing the Optimal Speeds Minimizing Energy Under Real-Time Constraints*. Research Report RR-9339. Inria Grenoble Rhône-Alpes, Apr. 2020. cit. on p. 17

[GGP20b] Bruno Gaujal, Alain Girault, and Stéphan Plassart. „Feasibility of on-line speed policies in real-time systems“. In: *Real-Time Systems* (Apr. 2020). cit. on p. 145

[GK03] 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: ACM, Aug. 2003, pp. 465–468. cit. on p. 82

[GN07] Bruno Gaujal and Nicolas Navet. „Dynamic voltage scaling under EDF revisited“. In: *Real-Time Systems* 37.1 (2007). The original publication is available at www.springerlink.com, pp. 77–97. cit. on p. 17

[GNW05] Bruno Gaujal, Nicolas Navet, and Cormac Walsh. „Shortest Path Algorithms for Real-Time Scheduling of FIFO tasks with Minimal Energy Use“. In: *ACM Transactions on Embedded Computing Systems (TECS)* 4.4 (2005). cit. on p. 44

[Gru01] 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*. 2001, pp. 11–16. cit. on pp. 42, 44

[Has10] Hado van Hasselt. „Double Q-learning“. In: *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada.* 2010, pp. 2613–2621.

[Hor74] W.A. Horn. „Some simple scheduling algorithms“. In: *Naval Research Logistics* 21.1 (1974), pp. 177–185. cit. on pp. 12, 13, 183

[HP91] Peter Hilton and Jean Pedersen. „Catalan Numbers, Their Generalization, and Their Uses“. In: *The Mathematical Intelligencer* 13.2 (Mar. 1991), pp. 64–75. cit. on p. 187

[IM95] Ilse Ipsen and Carl Meyer. „Uniform Stability Of Markov Chains“. In: *SIAM J. Matrix Anal. Appl.* 15 (Sept. 1995). cit. on p. 113

[JG04] R. Jejurikar and R.K. Gupta. „Procrastination Scheduling in Fixed Priority Real-Time Systems“. In: *Conference on Languages, Compilers, and Tools for Embedded Systems, LCTES'04*. Washington (DC), USA: ACM, June 2004, pp. 57–66. cit. on p. 147

[Li16] K. Li. „Energy and Time Constrained Task Scheduling on Multiprocessor Computers with Discrete Speed Levels“. In: *J. of Parallel and Distributed Computing* 95.C (Sept. 2016), pp. 15–28. cit. on p. 35

[LL73] C.L. Liu and J.W. Layland. „Scheduling Algorithms for Multiprogramming in Hard Real-Time Environnement“. In: *J. of the ACM* 20.1 (Jan. 1973), pp. 46–61. cit. on pp. 8, 148

[LS01] J.R. Lorch and A.J. Smith. „Improving Dynamic Voltage Scaling Algorithms with PACE“. In: *ACM Sigmetrics Conference*. 2001, pp. 50–61. cit. on pp. 42, 44, 81, 92

[LS04] J.R. Lorch and A.J. Smith. „PACE: A New Approach to Dynamic Voltage Scaling“. In: *IEEE Trans. Computers* 53.7 (2004), pp. 856–869. cit. on pp. 81, 82, 83, 94

[LY05] M. Li and F. F. Yao. „An efficient algorithm for computing optimal discrete voltage schedules“. In: *SIAM J. Comput.* 35 (2005), pp. 658–671. cit. on pp. 18, 81

[LYY17] Minming Li, Frances F. Yao, and Hao Yuan. „An $O(n^2)$ Algorithm for Computing Optimal Continuous Voltage Schedules“. In: *Annual Conference on Theory and Applications of Models of Computation, TAMC'17*. Vol. 10185. LNCS. Bern, Switzerland, Apr. 2017, pp. 389–400. cit. on pp. 17, 18, 33

[Mah96] Sridhar Mahadevan. „Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results“. In: *Machine Learning* 22.1-3 (1996), pp. 159–195.

[MCZ07] Jianfeng Mao, Christos G. Cassandras, and Qianchuan Zhao. „Optimal Dynamic Voltage Scaling in Energy-Limited Nonpreemptive Systems with Real-Time Constraints“. In: *IEEE Trans. Mob. Comput.* 6.6 (2007), pp. 678–688. cit. on pp. 18, 42

[MO79] Albert Marshall and Ingram Olkin. *Inequalitites: Theory of Majorization and Its Applications*. Vol. 143. Mathematics in Science and Engineering. Academic Press, 1979. cit. on pp. 54, 116

[MS02] Alfred Müller and Dietrich Stoyan. *Comparison Methods for Stochastic Models and Risks*. Wiley Series in Probability and Statistics ISBN: 978-0-471-49446-1. Wiley, 2002. cit. on p. 55

[ORR13] Ian Osband, Daniel Russo, and Benjamin Van Roy. „(More) Efficient Reinforcement Learning via Posterior Sampling“. In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. Ed. by Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger. 2013, pp. 3003–3011. cit. on p. 111

[OT96] DoKyeong Ok and Prasad Tadepalli. „Auto-Exploratory Average Reward Reinforcement Learning“. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 1*. 1996, pp. 881–887.

[Ous90] John K. Ousterhout. „Why Aren't Operating Systems Getting Faster As Fast as Hardware?“ In: *Proceedings of the Usenix Summer 1990 Technical Conference, Anaheim, California, USA, June 1990*. 1990, pp. 247–256.

[Pet01] M. Petrovitsch. „Sur une manière d'étendre le théorème de la moyence aux équations différentielles du premier ordre“. In: *Math. Ann.* 54.3 (1901), pp. 417–436. cit. on p. 179

[PS01] Padmanabhan Pillai and Kang G. Shin. „Real-time Dynamic Voltage Scaling for Low-power Embedded Operating Systems“. In: *SIGOPS Oper. Syst. Rev.* 35.5 (Oct. 2001), pp. 89–102. cit. on p. 42

[PT17] Doina Precup and Yee Whye Teh, eds. *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017. cit. on p. 142

[Put05] Martin L. Puterman. *Markov Decision Process : Discrete Stochastic Dynamic Programming*. Wiley Series in probability and statistics. Wiley, Feb. 2005. cit. on pp. 21, 42, 50, 51, 82, 91, 107, 135, 172

[RMM02] C. Rusu, R.G. Melhem, and D. Mossé. „Maximizing the System Value while Satisfying Time and Energy Constraints“. In: *Real-Time Systems Symposium, RTSS'02*. Austin (TX), USA: IEEE, Dec. 2002, pp. 246–255. cit. on p. 82

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. cit. on p. 171

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998. cit. on pp. 125, 130

[Sch93] Anton Schwartz. „A Reinforcement Learning Method for Maximizing Undiscounted Rewards“. In: *Machine Learning, Proceedings of the Tenth International Conference, University of Massachusetts, Amherst, MA, USA, June 27-29, 1993*. 1993, pp. 298–305.

[SPH07] Bernhard Schölkopf, John C. Platt, and Thomas Hofmann, eds. *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*. MIT Press, 2007.

[Sta+98] J. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Publisher, 1998. cit. on p. 17

[TCN00] L. Thiele, S. Chakraborty, and M. Naedele. „Real-Time Calculus for Scheduling Hard Real-Time Systems“. In: *International Symposium on Circuits and Systems, ISCAS'00*. IEEE, May 2000, pp. 101–104. cit. on p. 147

[WD92] Christopher J. C. H. Watkins and Peter Dayan. „Technical Note Q-Learning“. In: *Machine Learning* 8 (1992), pp. 279–292. cit. on pp. 125, 128, 130

[WRG16] J. Wang, P. Roop, and A. Girault. „Energy and Timing Aware Synchronous Programming“. In: *International Conference on Embedded Software, EMSOFT'16*. Pittsburgh (PA), USA: ACM, Oct. 2016. cit. on p. 35

[Wu+05] Q. Wu, P. Juang, M. Martonosi, and D.W. Clark. „Voltage and Frequency Control With Adaptive Reaction Time in Multiple-Clock-Domain Processors“. In: *International Conference on High-Performance Computer Architecture, HPCA'05*. San Francisco (CA), USA: IEEE, Feb. 2005, pp. 178–189. cit. on p. 35

[XMM05] R. Xu, D. Mossé, and R.G. Melhem. „Minimizing Expected Energy in Real-Time Embedded Systems“. In: *International Conference on Embedded Software, EMSOFT'05*. Jersey City (NJ), USA: ACM, Sept. 2005, pp. 251–254. cit. on pp. 92, 93

[XMM07] R. Xu, R.G. Melhem, and D. Mossé. „A Unified Practical Approach to Stochastic DVS Scheduling“. In: *International Conference on Embedded software, EMSOFT'07*. Salzburg, Austria: ACM, Sept. 2007, pp. 37–46. cit. on pp. 82, 92

[Xu+04] R. Xu, C. Xi, R. Melhem, and D. Mossé. „Practical PACE for Embedded Systems“. In: *International Conference on Embedded Software, EMSOFT'04*. Pisa, Italy: ACM, Sept. 2004, pp. 54–63. cit. on pp. 82, 92

[Yan+16] Shangdong Yang, Yang Gao, Bo An, Hao Wang, and Xingguo Chen. „Efficient Average Reward Reinforcement Learning Using Constant Shifting Values“. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.* 2016, pp. 2258–2264.

[YDS95] F. Frances Yao, Alan J. Demers, and Scott Shenker. „A Scheduling Model for Reduced CPU Energy“. In: *36th Annual Symposium on Foundations of Computer Science*. Milwaukee (WI), USA: IEEE Computer Society, Oct. 1995, pp. 374–382. cit. on pp. 4, 17, 23, 41, 42, 44, 53, 81, 92, 110, 115, 146, 153, 163

[YK03] H. Yun and J. Kim. „On energy-optimal voltage scheduling for fixed priority hard real-time systems“. In: *ACM Trans. Embed. Comput. Syst.* 2.3 (2003), pp. 393–430.

[Zha+05] Y. Zhang, Z. Lu, J. Lach, K. Skadron, and M.R. Stan. „Optimal Procrastinating Voltage Scheduling for Hard Real-Time Systems“. In: *Design Automation Conference, DAC'05*. San Diego (CA), USA: ACM, June 2005, pp. 905–908. cit. on p. 82

Additionally, the work done in this thesis comes partially from the following papers:

- Bruno Gaujal, Alain Girault, and Stephan Plassart. *Dynamic Speed Scaling Minimizing Expected Energy Consumption for Real-Time Tasks*. Tech. rep. hal-01615835. Inria, 2017.
- Bruno Gaujal, Alain Girault, and Stéphan Plassart. *A Discrete Time Markov Decision Process for Energy Minimization Under Deadline Constraints*. Research Report RR-9309. Grenoble Alpes ; Inria Grenoble Rhône-Alpes, Université de Grenoble, Dec. 2019.
- Bruno Gaujal, Alain Girault, and Stéphan Plassart. „A Linear Time Algorithm for Computing Off-line Speed Schedules Minimizing Energy Consumption“. In: *MSR 2019 - 12ème Colloque sur la Modélisation des Systèmes Réactifs*. Angers, France, Nov. 2019, pp. 1–14.
- Bruno Gaujal, Alain Girault, and Stéphan Plassart. *Exploiting Job Variability to Minimize Energy Consumption under Real-Time Constraints*. Research Report RR-9300. Inria Grenoble Rhône-Alpes, Université de Grenoble ; Université Grenoble - Alpes, Nov. 2019, 23p.
- Bruno Gaujal, Alain Girault, and Stéphan Plassart. *A Linear Time Algorithm Computing the Optimal Speeds Minimizing Energy Under Real-Time Constraints*. Research Report RR-9339. Inria Grenoble Rhône-Alpes, Apr. 2020.
- Bruno Gaujal, Alain Girault, and Stéphan Plassart. „Feasibility of on-line speed policies in real-time systems“. In: *Real-Time Systems* (Apr. 2020).