Dynamic Scheduling Strategies for Firm Semi-Periodic Real-Time Tasks

Yiqin Gao, Guillaume Pallez, Yves Robert, Frédéric Vivien

Dynamic Scheduling Strategies for Firm Semi-Periodic Real-Time Tasks

Yiqin Gao, Guillaume Pallez, Yves Robert and Frédéric Vivien

Abstract—This paper introduces and assesses novel strategies to schedule firm semi-periodic real-time tasks. Jobs are released periodically and have the same relative deadline. Job execution times obey an arbitrary probability distribution and can take either bounded or unbounded values. We investigate several optimization criteria, the most prominent being the *Deadline Miss Ratio* (DMR). All previous work uses some admission policies but never interrupt the execution of an admitted job before its deadline. On the contrary, we introduce three new control parameters to dynamically decide whether to interrupt a job at any given time. We derive a Markov model and use its stationary distribution to determine the best value of each control parameter. Finally we conduct an extensive simulation campaign with 16 different probability distributions. The results nicely demonstrate how the new strategies help improve system performance compared with traditional approaches. In particular, we show that (i) compared to pre-execution admission rules, the control parameters make significantly better decisions; (ii) specifically, the key control parameter is to upper bound the waiting time of each job; (iii) the best scheduling strategy decreases the DMR by up to 0.35 over traditional competitors.

Index Terms—real-time system, firm tasks, semi-periodic tasks, admission policy, job interruption, Markov model, scheduling strategy.

1 Introduction

This work focuses on scheduling techniques for firm semi-periodic real-time systems. Semi-periodic tasks [2], [3], [4], [5] have a variable execution time modeled by a Probability Distribution Function (PDF) and constant inter-arrival time. Firm tasks correspond to weakly-hard systems where some job deadlines can be missed [6]. Contrarily to soft tasks where all jobs must be completed eventually, there is no value to complete a firm job after its deadline. The main objective for firm tasks is then to minimize the *Deadline Miss Ratio* (DMR), i.e., the fraction of jobs that fail to complete execution before their deadline.

Firm real-time systems have many important applications, including multimedia, satellite-based tracking, financial forecast, and robotic assembly lines [7], [8]. A customary example is a video stream that requires frames to be played periodically, but the decoding/playing time of each frame is not constant. If the incoming rate of the frames is too high, or if their average processing time is too large, the server cannot successfully process all frames before their deadline, and the objective is indeed to decode as many frames as possible in due time, so as to maximize the quality of the output video.

To the best of our knowledge, few mechanisms have been proposed to minimize the DMR for firm tasks. All previous work relies on some admission policy, e.g., random, periodic, or with a bounded-size queue, but never interrupts a job once admitted, before it reaches its deadline. In other words, dynamic parameters that characterize the current load of the system are not taken into account. To give examples, at any given time, such parameters may include the duration of the current job so far, or the waiting time of the next job to execute. This work provides the design and evaluation of the first dynamic scheduling strategies for firm semi-periodic tasks. These new strategies apply to all execution scenarios but are shown particularly relevant when the system is saturated, meaning that many jobs cannot be processed successfully before their deadlines. In the case of a saturated system, we significantly decrease the DMR when compared to previous approaches.

The key novelty of our scheduler is the ability to interrupt the job currently executing on the server *at any time before its deadline*, in order to launch another job instead. This is a natural idea because a long-running job may put the following jobs at risk. However, when interrupting a job to launch a new one, the time already spent by the server to execute that job has been wasted, and there is no guarantee that the new job will complete faster than the interrupted one. Therefore, the decision to start executing a job but to interrupt it later before its deadline must be guided by sophisticated strategies. We investigate the impact of imposing constraints on three key control parameters: (i) the maximum completion time $d_{\max}$ before triggering interruption (e.g., which can be smaller than the deadline δ to favor next jobs); (ii) the maximal execution time $l_{\max}$ allotted to the job (e.g., which can depend upon the PDF of job execution times); and (iii) the maximum time elapsed between release and start-up $s_{\max}$ (e.g., which can be tuned according to system pressure). These control parameters provide much more flexibility than classic admission policies: in our framework, a job may be admitted but interrupted later, or even be never launched, while in previous approaches, it would proceed until completion or

A short version of this work has appeared as a 4-page Work-In-Progress paper at RTSS'2021 [1].

- *Yiqin Gao is with Shanghai Jiao Tong University, China. Yves Robert and Frédéric Vivien are with Laboratoire LIP, Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France. Guillaume Pallez is with Inria Bordeaux, France. Yves Robert is also with University of Tennessee Knoxville, USA. Contact: Yves.Robert@ens-lyon.fr.*

reaching its deadline, whichever comes first.

Our work focuses on a single periodic task and encompasses several new contributions:

- We deal with job execution times which obey an arbitrary probability distribution, with either bounded or unbounded support. As stated in the survey paper [9], a realistic model considers the jobs having varying execution times with given probability distributions. But most previous work assume a Worst-Case Execution Time (WCET) for jobs, while our dynamic scheduling mechanisms allow us to leverage this limitation;
- We consider several optimization objectives, some user-oriented, some platform-oriented: DMR, system utilization, mean response time for successful jobs, mean rejection time (see Section 3.1);
- We consider several job admission policies: all jobs, periodic, random, bounded-size queue (see Section 4.1);
- We introduce three new parameters to dynamically control scheduling strategies that take decisions on the fly, and that decide to execute a job, or not, based upon the current system state (see Section 4.2);
- We derive a Markov model and establish the existence of a unique stationary distribution for all scenarios, which we use to analytically determine the best value of the parameters and to express their performance (see Section 5);
- We conduct an extensive simulation campaign with 16 different probability distributions (and 12 out of 16 with several variants, either with unbounded support or truncated with a WCET) that lead to distinct system behaviors (see Sections 6 and 7).

Our main conclusions are: (i) compared to pre-execution admission rules, the control parameters make better decisions, however in some scenarios a well selected admission policy can improve some other objectives (such as response time) at almost no cost for the DMR; (ii) the key control parameter is to upper bound the waiting time of each job; (iii) the best scheduling strategy decreases the DMR by up to 0.35 over traditional competitors.

The rest of the paper is organized as follows. We survey related work in Section 2. We formally state the model and optimization criteria in Section 3. Section 4 is devoted to a description of the scheduling strategies. We introduce the Markov model and use its stationary distribution to compute the optimal values of the scheduling parameters in Section 5. We detail the experimental framework in Section 6 and present all results in Section 7. Finally, Section 8 gives concluding remarks and hints for future work.

2 Related Work

We survey related work in this section. For the presentation, we distinguish real-time applications (Section 2.1) from queuing networks (Section 2.2). This distinction is somewhat arbitrary, since similar ideas and techniques are used in both frameworks.

2.1 Real-Time Applications

The specification of real-time systems relies on their cyclic nature, and is expressed in terms of operations called jobs that have to be performed repetitively. Real-time jobs are released periodically and must complete successfully before a fixed time called the deadline. As already mentioned, semi-periodic tasks [2], [3], [4], [5] have a variable execution time which can be modeled by a PDF and constant inter-arrival time. When execution times cannot be assumed to be independent and identically distributed, some authors have proposed to model them via a Markov Chain [10]. See the survey by Davis and Cucu-Grosjean [11] for a more thorough overview.

In the literature, real-time systems are classified as *hard* and *weakly hard* [6]. For hard real-time tasks, no deadline should be missed: it is mandatory that each job completes before its deadline. On the contrary, for weakly-hard tasks, some deadlines can be missed. There are two sub-categories then: either a job that missed its deadline must still be completed (*soft* tasks), or it can be ignored (*firm* tasks). For soft tasks, it is acceptable to miss some deadlines occasionally, but all jobs must be completed eventually. For firm tasks, there is no value to complete a job after its deadline.

When a real-time system is saturated, it is unavoidable that some deadlines are missed. For firm tasks, the widely used (m, k) approach is to guarantee that at least m deadlines are satisfied out of every window of k consecutive jobs (and to have a ratio m/k as high as possible) [12], [13], [14], [15], [16][1]. For some applications, additional constraints, such as equally spacing the jobs matching their deadlines, may be enforced [17]. For instance, consider a video application where jobs are frames to process; the (m, k) approach with, say, $m = 10$ and $k = 20$, ensures that at least half the frames are delivered in time; but one might request that the good frames (those matching their deadlines) are more or less equally spaced in the window: receiving the first half of every window of 20 consecutive frames would damage the quality of the video! An interesting approach is explored in [18]: instead of aiming at DMR minimization, the goal is to derive the minimal server cost that guarantees a prescribed DMR value.

For soft tasks, the approach is different. All jobs have to be completed eventually. When the system is saturated, an approach is either to increase the deadline, or to decrease the release period, or both [19], [20]. Another approach is to accumulate a backlog of jobs that are waiting for completion and to analyze the expected response time of the system [21]. The system of [21] is analyzed using a Markov chain. We follow a similar Markov-based approach but focus on the job waiting times rather than on the backlog, because the backlog does not take job killing mechanisms into account (see the web supplementary material (WebSM) for details on the differences).

In addition, some work study a mixed firm/soft setting where late jobs (that miss their deadlines) still have some value. For instance three decisions (abort current job, skip next job, queue next job) are available to the scheduler in [22], [23], with the objective to minimize the DMR (see Section 3 for a definition). Finally, a variant of the problem

1. In this work, we do not assume that jobs always admit a finite worst-case execution time (WCET), or even that the duration of a job is always shorter than its relative deadline, which makes it impossible to guarantee any (m, k) constraint because some jobs can last for an arbitrary duration.

is dealt with in [24]: the i-th job J_i needs the output of job J_{i-1} to start, but can use that of job J_{i-2} if job J_{i-1} is late.

So far we have described systems where it was up to the scheduler to abort or skip some jobs. There are many variations: for instance, some jobs may have two different types, skippable or not [25], [26], and only skippable jobs (blue jobs in [25]) are allowed to miss their deadline.

2.2 Queuing Networks

In this work, we deal with the list of waiting jobs via a queue: while the machine is occupied, all submitted jobs are waiting for their turn (or waiting to be killed) in a queue. This strongly differs from what is done in queueing theory systems [27] where the typical constraint is to select in which order jobs should be executed; so as to optimize an objective such as response time. Furthermore, job execution times obey the same probability distribution; to select which jobs we execute next we use a simple First-Come-First-Served strategy amongst the jobs that have not been *rejected*.

The closer to our work is the job dropping model [28], where upon arrival of a job, the system selects, at admission, whether the job should be *dropped* (i.e., should be rejected). The decision to drop a job often depends on a function of queue parameters (number of jobs in the queue, average load of the queue): linear function (average queue size) [29] or other more complicated functions [30], [31]. Then, all jobs from the queue are executed (for example following a FCFS strategy). Contrarily to the job dropping model, we propose to drop jobs *dynamically*.

In a series of publications [32], [33], [34], [35], [36], job pruning techniques have been investigated. The authors consider an oversubscribed system to which jobs are submitted at random times. When a job is released, it is at first stored in a *batch queue*, and then can be allocated to the *machine queue* of a server by the mapping process. At each mapping event (completion or arrival of a job), the success probability of all jobs in the machine queue is recomputed, via a costly convolution over all possible durations of the jobs in the queue weighted by their respective probabilities. Jobs in the batch queue are also considered when there remains a server with available positions after the previous computation. Jobs with low probability to meet their deadline are dropped from the machine queue and deferred in the batch queue. Deferring jobs means that their assignment to a server is postponed. In contrast, a job dropped from the machine queue is definitely removed from the system. Contrarily to our problem, there are several job types, several server types, and job arrival dates are random rather than periodic. This calls for a very costly solution where a whole convolution over a large time window must be recomputed at each mapping event. Some jobs can be dropped after some duration, a strategy also investigated in this work. The striking difference is that with a single job type, and with periodic releases, we are able to determine the optimal value of the key scheduling parameters once and for all and to apply them on the fly, thereby providing a schedule whose cost is constant and independent of the number of jobs.

Finally, we note that saturated, or oversubscribed, systems become the norm in scientific computing. High-Performance Computing (HPC) systems run with a utilization of almost 100% and a job waiting time may span over several days [37]. In Cloud computing, the typical approach is dynamic pricing, with prices increasing as resources become more scarce [38]. In HPC, high waiting times are usually a deterrent with negative consequences, such as limiting the number of very large jobs on HPC platforms [37]. This work focuses on real-time systems, but borrows scheduling techniques from HPC and Cloud computing.

3 Model and Optimization Criteria

This section details the model and states all optimization criteria studied in this work.

3.1 Model and Optimization Criteria

We consider a single semi-periodic task [2], [3], [4], [5]. This task consists of jobs periodically input to the system. The execution time of each job follows a probability distribution $\mathcal{C}$. $\mathcal{C}$ can have either bounded support $[a, b]$ where $0 \leq a < b$ or unbounded support $[a, \infty]$ with $0 \leq a$.

The i^{th} job of the task, called job J_i, $i \geq 1$, is released at time $r_i = \tau \times (i-1)$ and placed into the waiting queue; it must complete not later than time $\delta_i = r_i + \delta$, where τ is the period and δ the relative deadline. If job J_i is not completed by its deadline δ_i, it is considered as failed, its execution is interrupted, and any time spent to execute part of it has been wasted.

Without loss of generality, we assume that the *arbitrary deadline* is such that $\delta > \tau$. Indeed, if $\delta \leq \tau$, the (trivial) solution with a single task is to never interrupt a job.

The problem becomes challenging when the system gets saturated: when we have, say, $\delta = 5\tau$: some jobs might execute for a time longer than τ, thereby delaying the beginning of the execution of the following jobs if they are not interrupted *before* reaching their deadline.

The main objective is to optimize the Deadline Miss Ratio (DMR) [21], [39]. This is the most frequently used metric in weakly-hard real-time systems, where one is looking for probabilistic guarantee that the deadline miss ratio of a job is below a threshold. Formally, the DMR when m jobs have been processed is the ratio $\frac{n(m)}{m}$, where $n(m)$ is the number of these m jobs that have completed successfully before their deadline. The asymptotic value of the DMR is defined as $\lim_{m\to\infty} \frac{n(m)}{m}$ if this limit exists. In Section 5.5, we show that this limit does exist for all the strategies that are considered in this work, and show how to compute its value using the Markov model.

In addition to the deadline miss ratio, we study the following three relevant criteria:

- *Utilization* [37]. This platform-oriented criterion corresponds to the proportion of time spent doing computation that ends up with a success (i.e., the completion of a job before its deadline). Note that the highest utilization does not guarantee the lowest deadline miss ratio, because the scheduler could decide to execute fewer longer jobs rather than more shorter ones.
- *Mean response time of successfully completed jobs* [21], [37]. This corresponds to the delay between the release of the job and its completion. When deadlines are enforced, this criterion is less important but does provide additional qualitative information about the various solutions.

• *Mean rejection time*. This new criterion is important for a saturated system. Intuitively, it corresponds to the time it takes for the system to inform a user that its job will not be executed. Of course, this criterion must be coupled with another, otherwise the best strategy would be to reject all jobs at submission time.

4 SCHEDULING

Our scheduling strategies rely on both admission/rejection policies, and on termination/interruption policies. An *admission policy* is the answer to the question: which jobs to admit into the system? Indeed, whenever a new job is released, the scheduler must decide right away whether the job is *admitted* into the system and granted the possibility to execute later. Alternatively, a job can be immediately *rejected* (without the possibility of later execution). Among the advantages of job rejection, one can envision: (i) a decrease of the load on the system; (ii) a decrease of the number of jobs that must be queued before execution; and (iii) a decrease of the time-to-failure for unsuccessful jobs.

The scheduler must then decide in which order to execute admitted jobs, whether to launch the execution of an admitted job or to cancel it, and when to *interrupt* a running job. In this last case, the job is considered failed (without the possibility of later execution). This is interesting when jobs are likely not to finish before their deadlines, in order to free the server for other jobs.

Because all jobs have the same relative deadline, we only consider policies which execute jobs in their order of arrival: these policies obey both the *first come first served* (FCFS) and the *earliest deadline first (EDF)* priority rules. We first cover possible admission policies (Section 4.1). Then, we describe a set of parameters controlling the launching and interrupting (killing) of jobs (Section 4.2). Finally, we define all the strategies under study (Section 4.3).

4.1 Admission Policies

We consider four types of admission policies:

1) The simplest admission policy is to admit all jobs submitted to the system. When needed, we tag variables describing the behavior under this admission policy with the tag ^{AA}, which stands for *All (jobs) Admitted*.
2) A classical admission policy is to use a bounded queue of a fixed size m. If, at the release time of job J_i there are already m jobs in the queue, job J_i is rejected. Strategies using a queue of size m will be tagged by the prefix QUEUE(m) and variables with the tag ^{queue}.
3) Random admission policy: each job submitted to the system will be admitted with probability α and rejected with probability $1-\alpha$ (regardless of the state of the system at the release time of the job). We mark variables with the tag ^{rand}.
4) Periodic admission policy. The admission policy is defined by a pattern (i.e., a boolean array) $\mathcal{A}$ of length $\tau_{\mathcal{A}}$. The job J_i submitted to the system will be admitted (regardless of the state of the system) if and only if $\mathcal{A}[i \bmod \tau_{\mathcal{A}}]$ is equal to TRUE. We mark variables with the tag ^{per}.

4.2 Control Parameters

Launching and interrupting jobs are controlled through the following three parameters (illustrated in Figure 1):

Upper bound on completion time: $d_{\max}$. All jobs have a relative deadline δ. Hence, a job released at time r will have its execution stopped at time $r+\delta$ if its execution has not successfully completed by that time. One can envision that for some distributions of job execution times, when nearing the absolute deadline $r+\delta$, the probability of the job completing successfully drops so much that it may not be worth to continue executing it until the deadline. To study this hypothesis, we introduce the upper bound $d_{\max}$ on the completion time of a job. Any job, released at time r, will then have its execution stopped at time $r+d_{\max}$ if it has not completed by that time. By definition of deadlines, any value for $d_{\max}$ greater than or equal to δ will have no influence on the system. Furthermore, if the value of $d_{\max}$ is (strictly) smaller than the period τ, one can prove by induction that the execution of any job will be completed or interrupted before the next job is released, and the server will be left idle. Hence, the range of meaningful values for $d_{\max}$ is: $\tau \leq d_{\max} \leq \delta$.

Upper bound on execution time: $l_{\max}$. One can envision long-tail distributions of job execution times for which long running jobs have very little probability to be successful while they waste a significant portion of the server time. Hence, the idea to have an upper bound $l_{\max}$ on the allowed execution time of jobs. Obviously, if $l_{\max} \geq d_{\max}$, this is equivalent to not having any upper bound on job execution times. On the other hand, if the value of $l_{\max}$ is (strictly) smaller than τ, the execution of any job will be completed or interrupted before the next job is released and the server will be left idle. Hence, the range of meaningful values for $l_{\max}$ is: $\tau \leq l_{\max} \leq d_{\max}$.

Upper bound on waiting time: $s_{\max}$. The longer a job has waited before its execution could start, the lower its probability of success. If a job has waited "too long" it may no longer be worth to launch it. Hence, the idea to set a relative upper bound $s_{\max}$ on the waiting time of jobs. With such a bound, any job J released at a time r will not be allowed to start its execution later than at time $r+s_{\max}$. Hence, if the server becomes available for job J only at time $r+s > r+s_{\max}$, then the job, although admitted to the system, will not be launched but discarded. The latest time at which the execution of the job preceding job J could complete or be interrupted is $(r-\tau)+d_{\max} = r+(d_{\max}-\tau)$. Therefore, any value for $s_{\max}$ greater than or equal to $d_{\max}-\tau$ will have no influence on the system. Hence, the range of meaningful values for $s_{\max}$ is: $0 \leq s_{\max} \leq d_{\max}-\tau$.

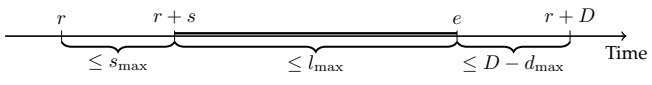


Fig. 1. A successful job released at time r whose relative deadline is δ, which starts its execution after s units of time, and that finishes at time e satisfies the constraints $s_{\max}$, $l_{\max}$, $d_{\max}$.

4.3 Scheduling Strategies

A scheduling strategy can now be defined entirely by an admission policy AP, and a triplet $(d_{\max}, l_{\max}, s_{\max})$. In

the following, we will denote by $d_{\max} = 4$ (for instance), a policy for which the value of $d_{\max}$ is set to 4 (with, of course, the same convention for $l_{\max}$ and $s_{\max}$).

AP-NEVERKILL. The simplest possible strategy is to let each job admitted with AP run until either its completion or it is killed by its deadline. We call this strategy NEVERKILL. It corresponds to AP, $(d_{\max}, l_{\max}, s_{\max}) = (\delta, \delta, \delta - \tau)$.

BUFFER(m). In this policy with a bounded queue, we only keep the most recent available jobs in the queue (i.e., when a new job arrives in the system, if the queue is full, then the oldest job from the queue is rejected). The intuition is that rather than rejecting the new job, it may seem wiser to reject the oldest one, because it has the smallest probability of success. This corresponds to the admission policy AA, along with the triplet $(d_{\max}, l_{\max}, s_{\max}) = (\delta, \delta, m\tau - 1)$: indeed, when $s_{\max} = m\tau - 1$, any job waiting at least a time $m\tau$ between its release time and the (potential) start of its execution is never launched. If a job has waited at least a time $m\tau$, it means that there are at least m other jobs waiting in the system.

AP-BESTDMAX, AP-BESTLMAX, and AP-BESTSMAX. A natural idea is to pick the value for $d_{\max}$ (resp. $l_{\max}$, $s_{\max}$) that minimizes the objective function when the other two parameters are non limiting. Non-limiting parameters are defined as $d_{\max} = \delta$, $l_{\max} = d_{\max}$, $s_{\max} = d_{\max} - \tau$. We discuss the complexity of these algorithms and their combination (optimizing for two or three parameters simultaneously) in Section 5.6.

In the following, when referring to a policy without mentioning the admission policy AP, we assume that AP $= AA$. The default objective is to minimize the DMR.

5 MARKOV MODEL

If the probability distribution $\mathcal{C}$ of job execution times is discrete, one can use a Markov chain to describe the evolution of the system under the NEVERKILL policy. Then one can compute its stationary distribution and evaluate its performance for the different objective functions expressed in Section 3. This is for instance what is done in [21] for an underloaded system.

In this section, we show how this approach can be applied to the continuous case, by using discretization, and how it can be extended to model all the admission policies listed in Section 4.1 and all the control parameters listed in Section 4.2. In Section 5.1, we start by describing the discretization (if $\mathcal{C}$ is initially continuous rather than discrete) and by setting notations. Then, in Section 5.2, we construct the Markov chain when all jobs are admitted into the system, before generalizing these Markov chains to other admission policies in Section 5.3. In Section 5.4, we prove that all these Markov chains admit a unique stationary distribution. In Section 5.5 we show how to express the four optimization criteria discussed in Section 3 in terms of the Markov model. Finally, in Section 5.6 we establish the complexity of using the Markov chains to compute the optimal values for the parameters $d_{\max}$, $l_{\max}$, and $s_{\max}$.

5.1 Discretization and Notations

We define a quantum duration q, and we use it to discretize all durations. We assume that the period τ and the relative deadline δ both last an integral number of quanta. In other words, we have $\tau = \lfloor \frac{\tau}{q} \rfloor q$ and $\delta = \lfloor \frac{\delta}{q} \rfloor q$.

To ease the writing of the equations, and up to the complexity analysis excluded (Section 5.6), we set that q lasts one arbitrary unit of time ($q = 1$ with the right choice of time unit), and that all durations are expressed in this arbitrary unit. Hence, the period now lasts τ quanta, and so on. The only assumption is that τ and δ are integers.

We approximate the probability distribution $\mathcal{C}$ of job execution times via discretization. Let f be the probability density function of $\mathcal{C}$. Let p_l be the probability that a job execution time is equal to l (quanta) in the discretized version of $\mathcal{C}$. We conservatively compute p_l by rounding up each job execution time to the nearest quantum multiple: $p_l = \int_{(l-1)q}^{lq} f(x)dx$.

We study the evolution of the system due to the execution of a job J released at a time r. Let s be the time job J has to wait after its arrival in the system until the server is available. Then the server will be available for job J at the time $r + s$. From Section 4.2, we have $0 \leq s \leq d_{\max} - \tau$. Also, the latest start time for a job is $s_{\max}$, and its execution can last at most a time $l_{\max}$. Hence, we also have $s \leq l_{\max} + s_{\max} - \tau$. Altogether, this gives us $0 \leq s \leq \sigma$ with $\sigma = \min\{s_{\max} + l_{\max}, d_{\max}\} - \tau$. Job J is followed by a job J' which is released at time $r + \tau$.

We will compute the probability $\mathcal{P}_{s,t}$ that the server is available at time $(r + \tau) + t$ for job J' if it was available at time $r + s$ for job J.

5.2 When All Jobs Are Admitted

In this section, we study systems in which all jobs are admitted (AA policy). Still, the execution of some jobs may be aborted because of the $l_{\max}$ and $d_{\max}$ parameters, and the execution of some jobs may not be launched at all because of the $s_{\max}$ parameter.

Let l denote the execution time of job J in the absence of any constraint (δ, $s_{\max}$, $l_{\max}$, and $d_{\max}$), and $\mathcal{P}_{t,t'}$ denote the probability that job J is available at t time units after release and is finished at t' time units after release. Without those constraints, the execution of job J starts at time $r + s$ and completes at time $r + s + l = (r + \tau) + (s + l - \tau)$. Therefore, $t = s + l - \tau$ (resp. $t = 0$ if $s + l - \tau \leq 0$: job J finished before the release time of job J'). Then, $\mathcal{P}_{s,t} = p_l$ (resp. $\mathcal{P}_{s,0} = \sum_{l \leq \tau - s} p_l$).

We now compute the value of $\mathcal{P}_{s,t}$ when taking $s_{\max}$, $l_{\max}$, and $d_{\max}$ into account. We first consider the case where the server is available too late for the job to be launched. Then we focus on the subcase $t = 0$.

- *Case* $s > s_{\max}$.
  In this case job J is not launched by definition of $s_{\max}$. Therefore the server becomes available for job J' at the time it became available for job J. Therefore, for any s and t such that $s_{\max} + 1 \leq s \leq \sigma$ and $0 \leq t \leq \sigma$:
  $$\begin{cases} \mathcal{P}_{s,\max\{0,s-\tau\}} = 1 \\ \mathcal{P}_{s,t} = 0 & \text{if } t \neq \max\{0, s - \tau\}. \end{cases}$$
- *Case* $0 \leq s \leq s_{\max}$ *and* $t = 0$.

The server is available at time $t = 0$ for job J' if the execution of job J lasted for a duration l' (after which it either completed or interrupted) such that $s+l'-\tau \leq 0$, that is, lasted for a duration l' such that $l' \leq \tau - s$. We first consider the subcase where the job execution completes. Then, the job execution length l must satisfy the two constraints: $l \leq l_{\max}$ and $s + l \leq d_{\max}$. Hence, $l \leq \min\{l_{\max}, d_{\max} - s\}$. Let $g(s) = \min\{l_{\max}, d_{\max} - s\}$. Then the job execution completes and leads to $t = 0$ if and only if: $1 \leq l \leq \min\{g(s), \tau - s\}$ (because in this case $l' = l$).

Moreover, the job execution is stopped and leads to $t = 0$ if and only if: $g(s)+1 \leq l$ and $l' \leq \tau - s$ with $l' = g(s)$. Altogether, we have:

$$\mathcal{P}_{s,0} = \sum_{l=1}^{\min\{g(s),\tau-s\}} p_l + \sum_{\substack{l \geq g(s)+1 \\ g(s) \leq \tau - s}} p_l$$

The second sum is null except when the only state is $s = 0$ ($l_{\max} = \tau$ or $d_{\max} = \tau$), in which case $\mathcal{P}_{0,0} = 1$.

- *Case* $0 \leq s \leq s_{\max}$ *and* $t \geq 1$.
  The constraint $t \geq 1$ can be fulfilled except if $\min\{s_{\max} + l_{\max}, d_{\max}\} - \tau \leq 0$, that is, except if the only possible state is $s = 0$. We first consider the subcase where the job execution completes. Then, the job execution length l must satisfy two constraints the two constraints: $l \leq l_{\max}$ and $s + l \leq d_{\max}$.
  Hence, $l \leq g(s)$ which is equivalent to $t \leq s - \tau + g(s)$. The case $t = s - \tau + g(s)$ contains both situations where the job execution completes and situations where the job execution is stopped. Hence, the following decomposition into subcases:
  - *Case* $1 \leq t \leq s - \tau + g(s) - 1$. We always have $l \geq 1$ which gives us the additional constraint: $t \geq 1+s-\tau$. In this subcase, the job completes after an execution which lasts $t - s + \tau$: $\mathcal{P}_{s,t} = p_{t-s+\tau}$.
  - *Case* $1 \leq t$ *and* $t = s - \tau + g(s)$. This corresponds to all remaining possible cases. Hence, $\mathcal{P}_{s,t} = 1 - \sum_{l=1}^{g(s)-1} p_l$.
  - *Case* $t > s - \tau + g(s)$. None of these cases can happen. Hence, $\mathcal{P}_{s,t} = 0$.

To illustrate the construction of the matrix $\mathcal{P}_{s,t}$, here is its value for a toy example where $\tau = 3$, $\delta = 8$, $d_{\max} = \delta$, $l_{\max} = 5$, and $s_{\max} = 4$:

$$\begin{bmatrix}
\sum_{i=1}^{3} p_i & p_4 & 1 - \sum_{i=1}^{4} p_i & 0 & 0 & 0 \\
\sum_{i=1}^{2} p_i & p_3 & p_4 & 1 - \sum_{i=1}^{4} p_i & 0 & 0 \\
p_1 & p_2 & p_3 & p_4 & 1 - \sum_{i=1}^{4} p_i & 0 \\
0 & p_1 & p_2 & p_3 & p_4 & 1 - \sum_{i=1}^{4} p_i \\
0 & 0 & p_1 & p_2 & p_3 & 1 - \sum_{i=1}^{3} p_i \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}$$

The last two rows correspond to cases where $s > s_{\max}$; in these cases job S is not launched and the processor becomes immediately available to job T. The first four rows contain the sum $1 - \sum_{i=1}^{4} p_i$ because $l_{\max} = 5$: if the execution time of S is strictly greater than 4 it uses the processor for 5 units of time after which either S successfully completes or it is killed because of $l_{\max}$. In the fifth row, job S waits for 4 units of time and, because $\delta = 8$, it uses the processor for at most 4 units of time; hence, the summation $1 - \sum_{i=1}^{3} p_i$. (The WebSM includes an example which shows the impact of the size of of the quantum on matrix $\mathcal{P}_{s,t}$.)

5.3 Other Admission Policies

Random admission policy. If the job J is not admitted, which happens with probability 1-α, then the server is available for job J' at time $\max\{0, s - \tau\}$. If the job J is admitted, which happens with probability α, the probability that the server is available for job J' at time t is the same as previously (under the condition that job J is admitted). Therefore, letting $\mathcal{P}^{rand}_{s,t}$ be the probability $\mathcal{P}_{s,t}$ for the random admission policy, we have:

$$\mathcal{P}^{rand}_{s,t} = \begin{cases} \alpha\mathcal{P}_{s,t} + (1 - \alpha) & \text{if } t = \max\{0, s - \tau\}, \\ \alpha\mathcal{P}_{s,t} & \text{otherwise.} \end{cases}$$

Periodic admission policy. This case is quite close to the previous one. The main difference is that we need to record where we are within the period. Let $\mathcal{P}^{per}_{(i-1)\sigma+s,t}$, with $1 \leq i \leq \tau_{\mathcal{A}}$, $0 \leq s, t \leq \sigma$, denote the probability that, if the server was available at time s for job J and if job J was the n-th job released in the system such that $1+(n \bmod \tau_{\mathcal{A}}) = i$, then the server will be available at time t for job J'. Then we have:

- Case $\mathcal{A}[i] = \text{TRUE}$ (job J is admitted).
$$\mathcal{P}^{per}_{(i-1)\sigma+s,t} = \mathcal{P}_{s,t}.$$
- Case $\mathcal{A}[i] = \text{FALSE}$.
$$\mathcal{P}^{per}_{(i-1)\sigma+s,t} = \begin{cases} 1 & \text{if } t = \max\{0, s - \tau\}, \\ 0 & \text{otherwise.} \end{cases}$$

Queues of size m. When all jobs were admitted, we were able to fully characterize the system for job J by solely relying on the time $r + s$ at which its processing would start. For queues, we need both this time and a vector $\vec{Q}$ of size γ (determined later) describing the state of the queue. For $0 \leq l \leq \gamma$, $\vec{Q}[l]$ is the number of jobs released *prior* to J which are still present in the system at the time $r+l\times\tau$, that is, the number of prior jobs which are either in the queue or being processed. Hence, components of $\vec{Q}$ takes value in $\{0, ..., m + 1\}$. Note that, because the execution of J starts at time $r + s$, the system does not contain any jobs released prior to J at least starting at time $r+s$. Therefore, $\vec{Q}[l] = 0$ if $l\times\tau \geq s$. By definition of σ, we have $s \leq \sigma$. Hence, $\gamma \leq \lfloor \frac{\sigma}{\tau} \rfloor$.

We need to compute the probability $\mathcal{P}^{queue}_{s,\vec{Q},t,\vec{R}}$ that the server is available at time $(r + \tau) + t$ for job J' with a queue state $\vec{R}$ if it was available at time $r + s$ for job J with queue state $\vec{Q}$. We start by characterizing the queue state if the server is available at time $r + t$ for job J'. We denote this queue state by $\vec{Q}^*$. We have two cases to consider:

- *Case* $\vec{Q}[0] = m + 1$. Then the queue is full and job J is not admitted. The only jobs released prior to J' that

may be present in the system are jobs released prior to J. Their presence is recorded in $\vec{Q}$. Hence:

$$\vec{Q^*}[l] = \begin{cases} 0 & \text{if } l\tau \geq t, \\ \vec{Q}[l+1] & \text{otherwise.} \end{cases}$$

The shift from $l+1$ to l is due to the fact that the reference point for job J' is one period (of length τ) later than for job J.

- *Case* $\vec{Q}[0] < m+1$. Then job J is admitted and is present in the system up to time t if $t > 0$. Hence:

$$\vec{Q^*}[l] = \begin{cases} 0 & \text{if } l\tau \geq t, \\ 1 + \vec{Q}[l+1] & \text{otherwise.} \end{cases}$$

If job J is admitted into the system, the probability that the server is available at time $r+t$ for job J' is $\mathcal{P}_{s,t}$. Indeed, once a job is admitted the behavior of the system is the same as when all jobs are admitted. If job J is not admitted, then the server is available for job J' at time $r+s$. Altogether, we obtain:

$$\mathcal{P}^{queue}_{s,\vec{Q},t,\vec{R}} = \begin{cases} \mathcal{P}_{s,t} & \text{if } \vec{Q}[0] < m+1 \text{ and } \vec{R} = \vec{Q^*}, \\ 1 & \text{if } \vec{Q}[0] = m+1,\ t = \max\{0, s-\tau\}, \\ & \text{ and } \vec{R} = \vec{Q^*}, \\ 0 & \text{otherwise.} \end{cases}$$

5.4 Asymptotic Behavior

In this section we prove that the Markov chains that we consider are both irreducible and aperiodic. Since their number of states is always finite, this is equivalent to say that these Markov chains are regular, hence, admit a unique stationary distribution [40].

Theorem 1. *The Markov chains of Sections 5.2 and 5.3 are all both irreducible and aperiodic, and they all admit a unique stationary distribution.*

Due to lack of space, the proof is available in the WebSM. A direct and important consequence of Theorem 1 is the following:

Corollary 1. *For any admission policy defined in Section 4.1 and any choice of the parameters $d_{\max}$, $l_{\max}$, and $s_{\max}$, the system converges to a unique asymptotic behavior.*

5.5 Optimization Criteria

This section explains how to express the four optimization criteria discussed in Section 3 in terms of the stationary distribution of Markov model. In the following let $\pi_\infty(s)$ denote the limit probability of occurrence of a state s, $0 \leq s \leq \sigma$ (we also have the limit probability $\pi_\infty(s,i)$ of state s at rank i in the pattern for the admission pattern policy, and $\pi_\infty(s,\vec{Q})$ for state s with queue state $\vec{Q}$ when admission is defined by a queue). We give the formula for all criteria with the AA policy, and refer to the WebSM for other admission policies.

- **Deadline miss ratio.** When admitting all jobs, we have

$$\mathbb{P}^{AA}_{succ} = \sum_{s=0}^{\sigma} \pi_\infty(s) \sum_{l=1}^{g(s)} p_l$$

and the DMR is equal to $\text{DMR}^{AA} = 1 - \mathbb{P}^{AA}_{succ}$.

- **Utilization.** The expectation of the time spent processing a successfully completed job is

$$\mathbb{E}^{AA}_{succ} = \sum_{s=0}^{\sigma} \pi_\infty(s) \sum_{l=1}^{g(s)} p_l l$$

and the utilization is the ratio $\mathbb{E}_{succ}/\tau$.

- **Mean response time of a successfully completed job.** This quantity is computed as

$$\sum_{s=0}^{\sigma} \pi_\infty(s) \sum_{l=1}^{g(s)} p_l (s+l)$$

- **Rejection time.** The expectation of the time spent processing an unsuccessful job is

$$\mathbb{E}^{AA}_1 \quad = \quad \sum_{s=0}^{\sigma} \pi_\infty(s) \left(\sum_{l \geq 1+g(s)} p_l \right) \quad \times \quad g(s)$$

while the expectation of the time spent waiting to start executing an unsuccessful job is

$$\mathbb{E}^{AA}_2 = \sum_{s=1+s_{\max}}^{\sigma} \pi_\infty(s) s_{\max}$$

The average rejection time is therefore $\mathbb{E}^{AA}_1 + \mathbb{E}^{AA}_2$.

5.6 Complexity

In this section, we establish the complexity of using the Markov chains to compute the optimal values of $d_{\max}$, $l_{\max}$ and $s_{\max}$ which minimize the deadline miss ratio (DMR).

Theorem 2. *The optimal value of $p \leq 3$ parameters (chosen from $d_{\max}$, $l_{\max}$, and $s_{\max}$) can be computed in time $O(\beta^3(\frac{\delta}{q})^p)$, where β is the number of states of the Markov chain. This holds both for the* DMR *and utilization criteria.*

See the WebSM for a proof of Theorem 2. We have $\beta \leq \frac{\delta-\tau}{q}$ when all jobs are admitted or when the admission is random, $\beta \leq \tau_{\mathcal{A}} \frac{\delta-\tau}{q}$ when jobs are admitted following a pattern of size $\tau_{\mathcal{A}}$ and $\beta \leq \frac{\delta-\tau}{q}(1+m)^\gamma = \frac{\delta-\tau}{q}(1+m)^{\frac{\delta}{\tau}}$ when jobs are admitted through a queue of size m. For instance, with the AA policy, the complexity becomes $\text{O}(\frac{\delta}{q})^6$ if we optimize for the three parameters simultaneously. In Section 7, we show that optimizing only for $s_{\max}$, and using a binary search instead of an exhaustive search, leads to very good results while reducing the complexity down to $\text{O}((\frac{\delta}{q})^3 \log(\frac{\delta}{q}))$.

6 Evaluation Methodology

In this section, we first discuss the evaluation protocol in Section 6.1. The analysis and computation of the solution with the Markov model relies on a discretization of time. The complicated problem to find a quantum size that ensures an accurate evaluation, while maintaining an acceptable complexity, is addressed in Section 6.2. Our code is publicly available at *https://perso.ens-lyon.fr/frederic.vivien/firm-semi-periodic* so that the interested reader can experiment with their favorite distribution.

6.1 Evaluation Protocol

In this section, we detail the experimental framework. As seen in Section 3.1, a periodic task is characterized by:

- the distribution $\mathcal{C}$ of job execution times.
- the period τ, which has direct impact on the system load. We consider $\tau \in [0.1E(\mathcal{C}); 2E(\mathcal{C})]$ by steps of 0.1; hence, 19 different values.
- the relative deadline δ. We select $\delta \in \{2\tau, 4\tau, 6\tau, 8\tau, 10\tau\}$. Intuitively, the ratio δ/τ is the number of consecutive jobs that can be simultaneously present in the system.

We use 12 standard probability distributions to generate job execution times (see the complete list and parameters in Table 1). In addition, multimodal distributions have been advocated to model jobs, file and object sizes [41]. Therefore, we also consider two types of bimodal distributions, either based upon truncated normal distributions or upon exponential distributions. For all these bimodal distributions, the two modes are equiprobable. Finally, for four distributions (gamma, log-normal, bimodal exponential, and bimodal truncated normal), we consider two different values for the standard deviation, in order to illustrate different potential behaviors. Altogether, we have 16 distributions. To enable a direct comparison, we choose their parameters so that they all achieve an expectation $E(\mathcal{C}) = 1$.

TABLE 1
Parameters for the distributions used in the simulations. The parameter nomenclature follows the habitual definitions (for instance, μ is the mean, σ the standard deviation, except for truncated and half-normal distributions where μ and σ are the mean and standard deviation of the original normal distribution).

Distribution	Parametrization
Bimodal exponential	$\mu_1 = 1.005, \mu_2 = 0.995$
	$\mu_1 = 0.1, \mu_2 = 1.9$
Bimodal truncated normal	$\mu_1 = 0.5, \sigma_1 \approx 0.534, \mu_2 = 1, \sigma_2 \approx 1.068$
	$\mu_1 = 0.01, \sigma_1 \approx 0.178, \mu_2 = 1, \sigma_2 \approx 1.782$
Exponential	$\mu = 1$
Gamma	$k = \frac{1}{3} \approx 0.333, \theta = 3$
Half-normal	$\sigma = \sqrt{\frac{\pi}{2}} \approx 1.253$
Inverse Gamma	$\alpha = \frac{7}{3} \approx 2.333, \beta = \frac{4}{3} \approx 1.333$
Log-normal	$\mu = 1, \sigma = 0.5$
	$\mu = 1, \sigma = 3$
Truncated normal	$\mu = 0.8, \sigma \approx 0.754$
Uniform	$a = 0, b = 2$
Weibull	$k \approx 0.411, \lambda = \frac{1}{\Gamma(1+\frac{1}{k})} \approx 0.324$
	$k = 1.5, \lambda = \frac{1}{\Gamma(1+\frac{1}{k})} \approx 1.108$
Gumbel	scale ≈ 0.0945, location = $10\times$ scale
Beta	$\alpha = 1.5, \beta = 4$

Reduced evaluation set

In the experiments, we observed that many distributions had similar behaviors. Similarly, the behavior was quite consistent when the ratio $N = \delta/\tau$ changed. For readability and interpretability of the results, in the rest of this paper, we report only a subset of the results: (i) we only present $N \in \{2, 6, 10\}$; (ii) we only keep five distributions, those of the first column of Table 2 which categorizes the 16 distributions (we keep two distributions per generic behavior and the Gumbel distribution whose behavior is quite unique). Complete results for all distributions and values of N are available in the WebSM.

TABLE 2
Categorizing the behavior of the 16 distributions.

Representative distribution	**Distributions with similar behavior**
LogNormal $\sigma = 3$ (LN(3)); Exponential (EXP)	Bimodal Exponential (both); Weibull $k < 1$; Bimodal truncated normal ($\mu_1 = \mu_2/100$)
LogNormal $\sigma = 0.5$ (LN(0.5)); Truncated Normal (TRUNCNORM)	Uniform; Half-normal; Weibull $k > 1$; Bimodal truncated normal ($\mu_1 = \mu_2/2$); Gamma; Inverse Gamma; Beta
Gumbel	Gumbel

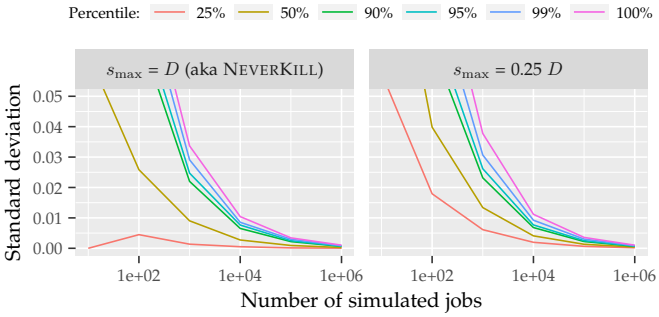


Fig. 2. Evolution of the standard deviation of a set of 100 simulations for different $s_{\max}$ heuristics, as a function of the number of simulated jobs. For each number of jobs, we report percentiles on the standard deviation of the DMR for different periods, deadlines, and distributions.

6.2 Accuracy

In order to evaluate the different scheduling algorithms, we have created an event-based simulator. This simulator takes as input the application model (probability distribution, period, deadline) and a number of events $\mathcal{N}$. It starts from an empty queue and simulates an execution until $\mathcal{N}$ jobs have been submitted into the system. The objectives are then evaluated over the complete execution.

Instantiation of the event-based simulator

The first question is how many events $\mathcal{N}$ should be chosen for the evaluation? In order to answer it, we perform the following experiments. Consider the algorithm S-MAX : $s \mapsto (AA, (d_{\max}, l_{\max}, s_{\max}) = (\delta, \delta, s))$ (using the notations from the Section 4.3, this means that we accept all jobs, $d_{\max}$ and $l_{\max}$ are not constraining, and $s_{\max}$ is the input of the algorithm): by construction s belongs necessarily to the set $[0, \delta]$. We consider four variants of it: S-MAX$(x\delta)$ for $x \in \{0.25, 0.5, 0.75, 1\}$ to cover the whole scope of values.

When $\mathcal{N}$ varies in $\{10^1, 10^2, 10^3, \ldots, 10^6\}$:

- We repeat all experiments proposed in Section 6.1 ($16 \times 19 \times 5 = 1520$ experiments), 100 times.
- We measure the standard deviation of all scenarios (1520 data points).
- We plot some quantiles of these standard deviations as a function of $\mathcal{N}$. These results are partially presented in Figure 2 (the other results are in Figure 1 of the WebSM).

In all instance, the standard deviation with 10^6 jobs is lower than 0.0011. This indicates that we can rely with high confidence on a single experimental evaluation with

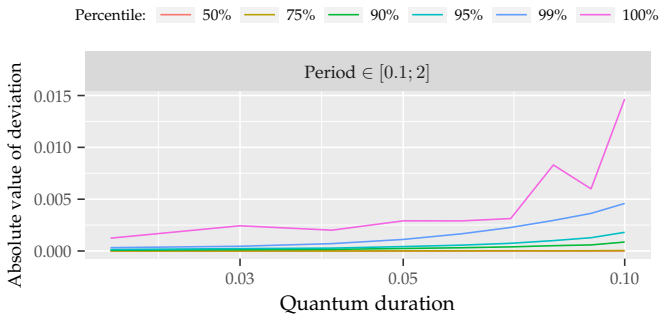


Fig. 3. Absolute value of deviation from the performance simulated with 10^6 jobs predicted for heuristic BESTSMAX whose parameter $s_{\max}$ was defined by a Markov chain using a quantum $q = 0.01$. For each quantum size, many different periods, deadlines, and distributions are evaluated whose percentiles (and worst-case) are presented.

$\mathcal{N} = 10^6$ events. The time to execute a single evaluation with 10^6 jobs is 0.01 second on a laptop.

Discretization quantum

In Section 4.3, we presented algorithms which are defined by the optimal choice for the parameters $d_{\max}$, $l_{\max}$ and/or $s_{\max}$ (e.g., AP-BESTDMAX). All these algorithms rely on computing the stationary distribution of a Markov Chain, based on a model that discretizes time into quanta of size q (minimum time unit). The smaller the quantum, the more accurate the solution and evaluation. However, the complexity of the algorithms increases with the quantum size (Theorem 2). The next question is the maximum size of q for which the performance of the algorithms remains close to optimality in real and continuous settings

In order to find this value, we consider the BESTSMAX algorithm for AA policy. For a quantum size q, we denote by BESTSMAX(q) this algorithm. We study the performance of $q \mapsto$ BESTSMAX(q) when $q \in [0.01, 0.1]$ for all scenarios detailed in Section 6.1. Then, in Figure 3, we plot the absolute value of the deviation from the most precise model:

$$q \mapsto |\text{DMR}(\text{BESTSMAX}(q)) - \text{DMR}(\text{BESTSMAX}(0.01))|$$

that we evaluate with $\mathcal{N} = 10^6$ events.

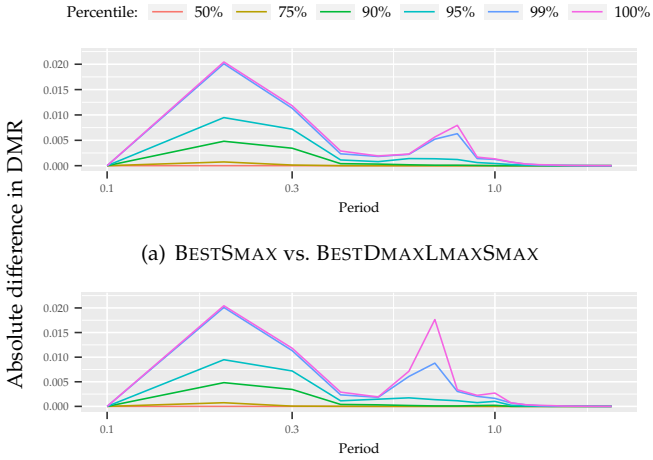
From Figure 3, we see that the deviation is quite stable. The performance of the model is accurate with $q = 0.1$. Indeed, for the case where $q = 0.1$, the deviation from the solution with $q = 0.01$ is always smaller than 0.015. Hence, in the following, we use $q = 0.1$ for the evaluations.

7 EVALUATION RESULTS

In this section, we evaluate the performance of the scheduling strategies. In Section 7.1, we focus on admission policy AA (all jobs are admitted) and study the impact of the three parameters $d_{\max}$, $l_{\max}$, and $s_{\max}$ on the DMR. Then, in Section 7.2 we compare the performance achieved for AA with the other admission policies. Finally, in Section 7.3, we discuss the impact of the different admission policies on objectives other than DMR (see Section 3.1).

7.1 When All Jobs Are Admitted

In Section 4.2 we have introduced three execution control parameters, namely $d_{\max}$, $l_{\max}$ and $s_{\max}$. The first question to answer is: **what is the relative impact of these parameters on the deadline miss ratio?** To answer this question, we compare on Figure 4 absolute performance in terms of DMR of different heuristics.



(a) BESTSMAX vs. BESTDMAXLMAXSMAX

(b) BESTDMAXLMAX vs. NEVERKILL

Fig. 4. Absolute difference in the simulated performance between the named heuristics. For each value of the period τ, the main percentiles (and worst-case) are presented.

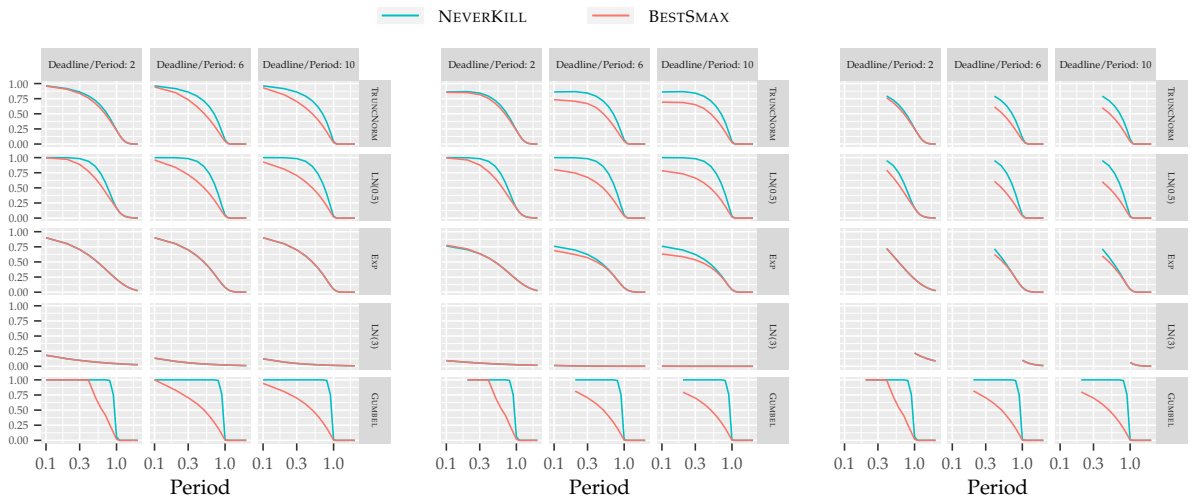
The most complete heuristic is the BESTDMAXLMAXSMAX heuristic which chooses the combination minimizing the DMR for a given period τ, a given distribution $\mathcal{C}$, and a given relative deadline δ, among all the possible values for the three parameters. Hence, we use BESTDMAXLMAXSMAX as a reference. On Figure 4(a) we compare the performance of BESTSMAX and BESTDMAXLMAXSMAX. We observe that the difference of their absolute performance over all distributions is minimal: the worst case is reached with an absolute difference of 0.021. This means that using the parameters $d_{\max}$ and $l_{\max}$ on top of parameter $s_{\max}$ in the most favorable case only leads to a decrease of 0.021 of the DMR, which is not significant.

We confirm this observation in Figure 4(b), where we compare the performance of BESTDMAXLMAX and NEVERKILL. In other words, we compare the strategy that optimizes $d_{\max}$ and $l_{\max}$, while $s_{\max}$ is not constraining, with the strategy where none of the parameters are constraining, i.e., that let each job run until either completion or interruption at deadline. Again, the absolute difference of performance is always below 0.021.

This further confirms that parameters $d_{\max}$ and $l_{\max}$ play no significant role in optimizing the DMR.

Therefore, in the remainder of this study of the AA admission policy, we focus on the BESTSMAX heuristic and the impact of the $s_{\max}$ parameter. We compare it to the NEVERKILL algorithm which corresponds to setting $s_{\max} = \delta - \tau$ (not constraining).

The second question to answer is: **when do we need to compute the $s_{\max}$ parameter?**



(a) Distributions as defined in Table 1.

(b) The same distributions truncated with a Worst-Case Execution Time of 6τ.

(c) These distributions scaled to have a mean of 1 when truncated with a WCET of 6τ.

Fig. 5. Absolute performance of heuristics BestSmax and NeverKill (lower is better) with different distributions.

Figure 5(a) displays the DMR of BestSmax and NeverKill while varying the period τ, the relative deadline δ, and the distribution of job execution times $\mathcal{C}$. The lower the period, the more loaded the system, the higher the DMR for both strategies. The main observation is that we have two typical behaviors:

1) Distributions where the optimal solution is to not use $s_{\max}$ as a constraint (BestSmax = NeverKill), this is the case for the category of distribution represented by Exp and LN(3);
2) Distributions where the value of $s_{\max}$ is important (TruncNorm, LN(0.5), and Gumbel). The peak performance are under heavy load ($\tau < 1$) and large deadlines.

The performance of NeverKill does not appear to be impacted by the value of the relative deadline (δ) with respect to the period (τ). On the contrary, the improvement of BestSmax with respect to NeverKill increases with the ratio $\frac{\delta}{\tau}$ and appears to reach a plateau when $\frac{\delta}{\tau} = 4$ or 6, depending on the distributions.

We can look specifically at the value of $s_{\max}$ returned by the BestSmax algorithm. In Figure 6, we represent these values, normalized to $\delta - \tau$. In practice, the performance of BestSmax can be similar for very different values of $s_{\max}$. On the one hand, this makes BestSmax rather robust. On the other hand, this explains the fact that some curves are not smooth on Figure 6. The discretization also plays a role in these irregularities. This figure confirms the two categories found in the previous result.

Altogether, we have shown that using BestSmax algorithm can have a significant impact for some distributions. For other distributions, $s_{\max}$ has no impact (just like $d_{\max}$ and $l_{\max}$), and NeverKill is a better solution: it has same performance but does not need any pre-computation.

Understanding what makes a distribution a good candidate for either one of the categories is a challenge that we leave to the future.

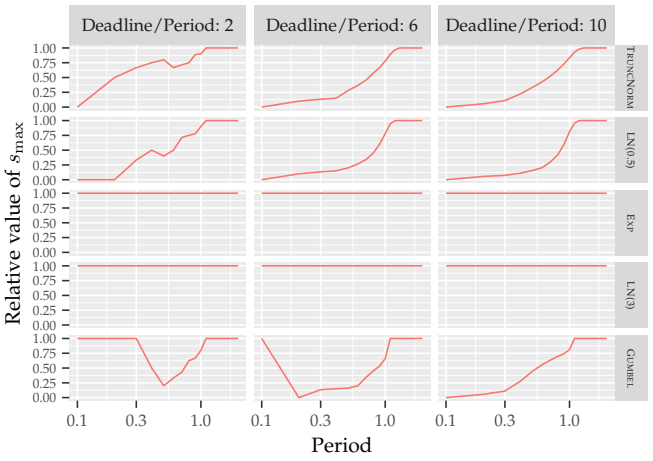


Fig. 6. Best value for the $s_{\max}$ threshold for heuristic BestSmax expressed as a fraction of the maximum meaningful value $\delta - \tau$.

One limitation of the BestSmax heuristic is its computational complexity. The third question to ask is: **can we find an efficient strategy for finding a good $s_{\max}$ value?** A natural strategy for guessing the optimal $s_{\max}$ value is to study the Buffer(m) strategies. Indeed, as already mentioned in Section 4.3, Buffer(m) corresponds to a particular choice for $s_{\max}$, since this strategy is defined by $(d_{\max}, l_{\max}, s_{\max}) = (\delta, \delta, m\tau - 1)$. Figure 7 compares the DMR achieved by Buffer(m) and BestSmax. By definition, Buffer(m) can never achieve a better DMR than BestSmax. However, is there a value of m for which the performance is always similar? It does not appear to be the case. On the one hand, the larger the period, the larger the buffer size, the better the DMR. On the other hand, lower

values for m achieve better performance for lower periods (saturated systems) for most distributions, but not for a few distributions like LN(3). Therefore, BUFFER(m) strategies do not appear to be a good substitute for BESTSMAX.

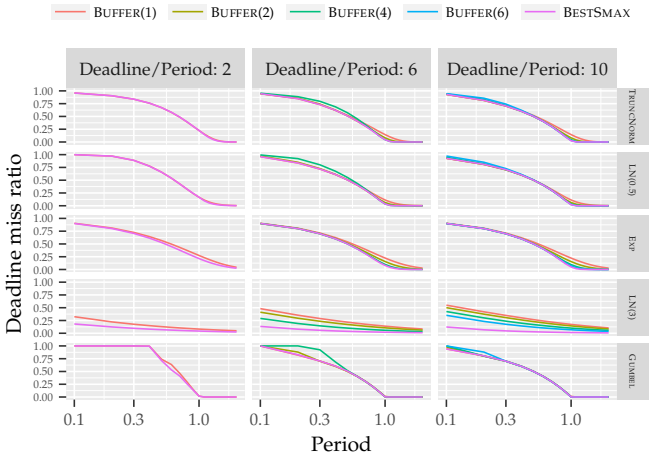


Fig. 7. Deadline miss ratio of heuristics BESTSMAX and BUFFER(m) for different period τ and distributions while varying q.

Still, is it necessary to test all the $1 + \frac{\delta\tau}{q}$ possible values of $s_{\max}$ to define BESTSMAX? We conjecture that the DMR of the function S-MAX : $s \mapsto (AA, (d_{\max}, l_{\max}, s_{\max}) = (\delta, \delta, s))$ should be rather regular and have a single optimum. In other words, we believe that this function should be either increasing, or increasing and then decreasing. Although we have not been able to prove this conjecture, we have tested it.

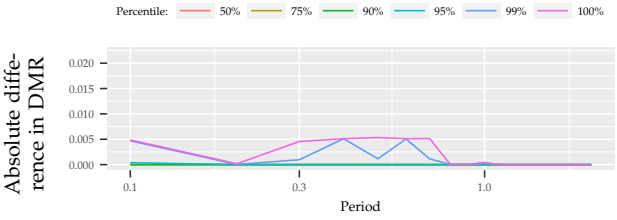


Fig. 8. Absolute difference in the simulated performance between BESTSMAX vs. BINSMAX. For each value of the period τ, the main percentiles (and worst-case) are presented.

We propose the BINSMAX heuristic: a variant of BESTSMAX which uses a binary search, instead of an exhaustive one, to determine its choice of value for $s_{\max}$. On Figure 8 we compare the DMR of BESTSMAX and BINSMAX. The difference is always smaller than 0.006. Hence, instead of using BESTSMAX, one can use BINSMAX which achieves the same performance but which is less expensive to define.

In conclusion, we have shown that for the AA policy, one can focus on the BESTSMAX algorithm. For some distributions it performs extremely well compared to the natural NEVERKILL policy, while for others it is more interesting to use NEVERKILL which has a much lower computational complexity and achieves the same performance. Using BESTSMAX is particularly important for saturated systems

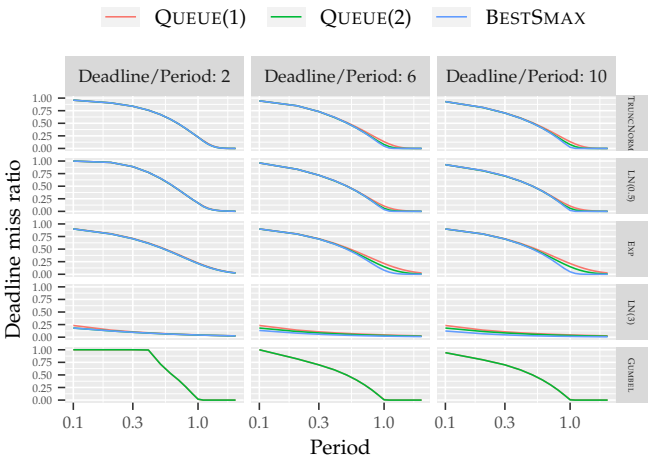


Fig. 9. Deadline miss ratio of job admission policy QUEUE(m) applied on BESTDMAXLMAXSMAX for different period τ and distributions.

with long deadlines, the DMR can be decreased by up to 0.35.

Because all but two of the studied distributions have unbounded support, one may wonder whether this property impacts the performance of scheduling strategies and of our conclusions. To answer this question we present on Figure 5 the performance of NEVERKILL and BESTSMAX for the distributions described in Table 1 (Figure 5(a)), when these distributions are truncated so that the Worst-Case Execution Time (WCET) of jobs is 6τ (Figure 5(b)), and when these distributions are first scaled and then truncated, so that the average execution time of jobs is equal to 1 and the WCET is 6τ (Figure 5(c)). The choice of the value 6τ for the WCET enables to consider cases where the relative deadline is smaller, equal, or greater than the WCET. The missing parts of the graphs on Figure 5(c) (and for Gumbel on Figure 5(b)) correspond to parameter settings for which there does not exist a distribution satisfying all the constraints.

One can see that when job execution times are bounded BESTSMAX achieves even better performance when compared to NEVERKILL: for distributions for which BESTSMAX already achieved lower DMRs than NEVERKILL the difference in DMRs is larger; for some distributions (like exponential) for which the two strategies achieved the same DMR, BESTSMAX now outperforms NEVERKILL when the system is saturated.

7.2 Other Admission Policies

In this Section, we discuss other admission policies. Their main difference with AA is that they can choose to reject a job at submission time. For each of the three policies AP which do not admit all jobs (random, periodic and with queues), we compare the performance of AP-BESTDMAXLMAXSMAX, with the policy from the previous section AA-BESTSMAX (simply denoted BESTSMAX).

We start by studying queues with Figure 9. Note that an AA policy corresponds to a queue of unbounded size (although the queue is necessarily bounded by δ/τ). Whatever the distribution and set of parameters, QUEUE(m) never achieves better performance than AA.

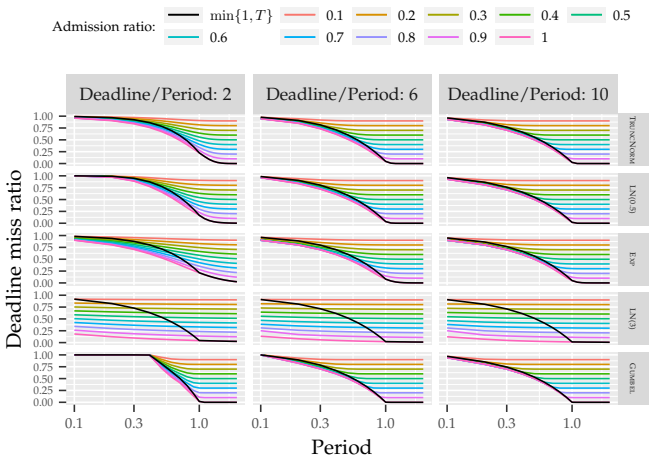


Fig. 10. Deadline miss ratio of random job admission policy applied on BESTDMAXLMAXSMAX for different period τ and distributions.

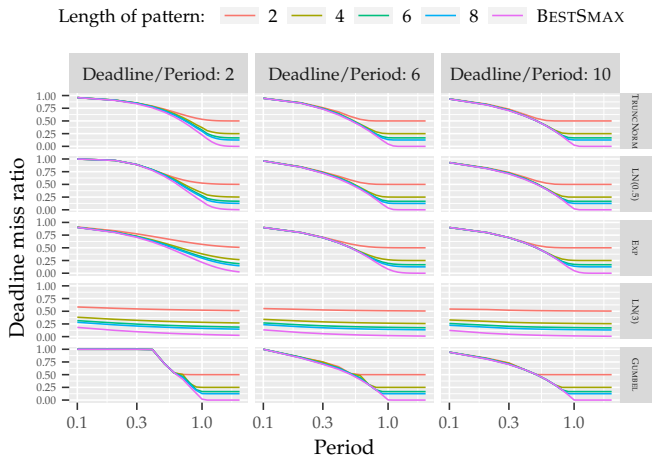


Fig. 11. Deadline miss ratio of periodic job admission policy applied on BESTDMAXLMAXSMAX for different period τ and distributions.

A striking observation however, is that under heavy load, the performance of QUEUE(1)-BESTDMAXLMAXSMAX is quite close to that of the AA policy, with guarantees that the number of jobs waiting is quite small. This is an interesting result as it allows to give faster negative answers to users when we know that it is very likely that their jobs will not be accepted.

There is an exception for some distributions that correspond to the category of distributions where BESTSMAX = NEVERKILL (such as LN(3)). In this case, QUEUE(m) achieves worse performance even for very small periods.

Overall, bounded queues are detrimental to the performance. This is particularly true for distributions where the optimal strategy under AA is NEVERKILL. However for other distributions and under heavy load, the performance loss when using queues of size 1 is negligible.

For the random admission policy, Figure 10 presents the DMR of BESTDMAXLMAXSMAX for different values of the admission ratio. Whatever the distribution and set of parameters, the higher the admission ratio, the lower the DMR. The best choice is to have an admission ratio of 1, that is, to admit all jobs. Again we observe significant difference for the category of distributions where NEVERKILL is the best strategy for the AA policy. One could wonder whether one could find an admission rate that depends on the load of the system that could be efficient for the other categories of distributions. A natural heuristic is one where the admission ratio is equal to the period, hence, ensuring a load of 1. This is plotted in black. This strategy does not perform well for any distribution.

Figure 11 presents the DMR for periodic admission patterns. For each pattern length under study, we generate all possible patterns of that length which do not admit all jobs and pick the one leading to the lowest DMR. The longer the pattern, the lower the DMR. This is easily explained: in each case the best pattern admits all jobs except one.

Overall, to minimize the DMR, it is always better to admit all jobs and then to control their execution, than to reject some jobs at submission time.

7.3 Performance for Different Optimization Criteria

So far, we have assessed the performance of the different scheduling strategies and admission policies when trying to minimize the DMR. In this case, the AA policy is dominant, there is no incentive to reject jobs early on. In this section we study secondary optimization criteria (defined in Section 3.1) to see if it may be interesting from another perspective to have different admission policies.

In Figure 12, we report the performance for the four criteria, for three of the admission policies. We report the performance of BESTDMAXLMAXSMAX (for DMR minimization) either with a queue of size 1 (QUEUE(1)) or with a random admission policy with an admission ratio equal to the period (RAND($\min\{1, \tau\}$) (recall that $\tau \in [0.1, 2]$ hence the need for the minimum)). When all jobs are accepted, we report the performance of BESTSMAX (which minimizes DMR). Finally, we also include the performance of a strategy that uses the set of $d_{\max}$, $l_{\max}$, and $s_{\max}$ parameters which maximizes the utilization (BESTDMAXLMAXSMAXUTIL).

As seen in the previous sections, except for RAND($\min\{1, \tau\}$), the performance of the heuristics is quite close for DMR minimization. For utilization, however, the heuristics achieve significantly different performance, especially for the exponential distribution. This hints at different possible trade-offs between DMR and utilization for those scenarios where utilization would be a criterion of interest.

For both the rejection time and the response time, BESTSMAX and BESTDMAXLMAXSMAXUTIL achieve worse performance than QUEUE(1). This can be easily explained. QUEUE(1) rejects many jobs at their admission time. This decreases the load in the system and lower the probability that jobs are killed by their deadlines which, in turn, decreases the rejection time. This is especially true when the period is greater than 1. In such a case, the system is underloaded and rejected jobs are dropped exactly at their deadlines. Also, when the system is less loaded, jobs are less impacted by their predecessors, which decreases their waiting time, and thus their response time. RAND($\min\{1, \tau\}$) has an interme-

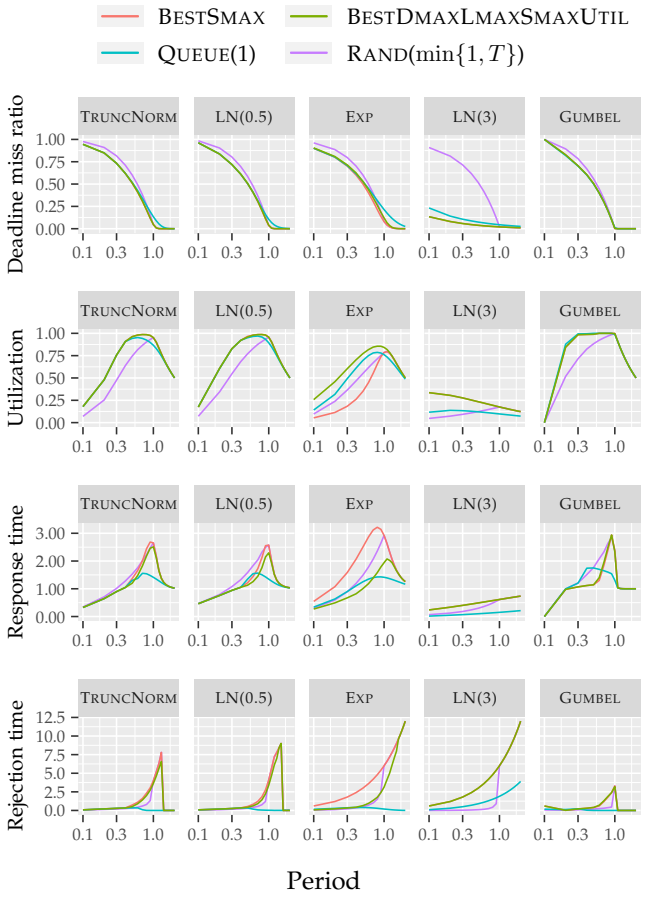


Fig. 12. Performance of different optimization criteria for different heuristics, while varying period τ and fixing $N = 4$.

diate behavior due to a higher acceptance rate.

7.4 Summary

Our thorough evaluation provides several important insights about scheduling strategies for saturated real-time systems. The first one is that the distribution of job execution times matters a lot: for some distributions, the naive strategy NEVERKILL (which admits all jobs and let them run until either they finish or meet their deadline) performs very well. For other distributions, the best strategy is to accept all jobs, and to decide early termination only based on the time spent in the queue (parameter $s_{\max}$). Determining whether a distribution fits either category remains an open problem.

Finally, we investigated how optimizing DMR tradesoff with other optimization criteria. While BESTSMAX dominates in performance for DMR and utilization, it comes at the price of higher response time and rejection time. When quality of service is important, intermediate admission policies (such as QUEUE(1)) can be used under heavy load, and with a negligible increase of DMR.

8 CONCLUSION

This work has discussed several novel strategies to schedule firm semi-periodic jobs. Previous work was limited to the NEVERKILL strategy coupled with some admission policy. We have introduced three control parameters to dynamically decide when to start or interrupt a job. Aiming at different optimization criteria, our dynamic scheduling strategies encompass a wide range of execution scenarios.

We have used discretization to derive a Markov model and used its stationary distribution to determine the best value for each control parameter. An extensive simulation campaign with 16 different probability distributions has shown that tuning the new control parameters, in particular $s_{\max}$, dramatically improves system performance. We have reduced the complexity to find the optimal value of $s_{\max}$ by using a binary search. The ultimate goal would be to identify a priori for which distributions $s_{\max}$ is necessary, and for which ones NEVERKILL should be preferred.

This work has laid the foundations for the study of dynamic scheduling strategies with a single periodic task and a single server. With $p \geq 2$ processors, a first approach could use a round-robin allocation of jobs to the p processors, which amounts to solving the problem with a single processor and an arrival rate divided by p. Another approach for p processors would be to use earliest start times to assign incoming jobs, but this would require a Markov chain with a huge number of states, so it might not be feasible in practice. As for several periodic tasks, this is a very challenging problem. The job execution times of different tasks will have different probability distributions and may have very different means (different granularities). One would need to use some priority to choose which tasks to favor, or to enforce some fairness criterion, such as minimizing the maximum DMR over all tasks. We plan to design some strategies based upon dynamic control parameters that would extend our work for a single task. The Markov chains involved would also have a huge number of states.

The main focus of this work has been on the DMR objective. Future work will further study system utilization, which is an important criterion for platform owners. Combining several objectives may be important too, e.g., minimizing DMR or maximizing utilization, while enforcing an upper bound on mean rejection time.

Acknowledgments

We would like to thank the reviewers for their comments and suggestions, which greatly helped improve the final version of the paper.

REFERENCES

[1] Y. Gao, G. Pallez, Y. Robert, and F. Vivien, "Evaluating Task Dropping Strategies for Overloaded Real-Time Systems (Work-In-Progress)," in *42nd Real Time Systems Symp. (RTSS)*. IEEE, 2021.
[2] L. Abeni and G. Buttazzo, "Stochastic analysis of a reservation based system," in *IPDPS*. IEEE, 2001, pp. 946–952.
[3] ——, "QoS guarantee using probabilistic deadlines," in *Proc. 11th Euromicro Conf. Real-Time Systems*. IEEE, 1999, pp. 242–249.
[4] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J.-S. Liu, "Probabilistic performance guarantee for real-time tasks with varying computation times," in *Proceedings Real-Time Technology and Applications Symposium*. IEEE, 1995, pp. 164–173.
[5] A. Atlas and A. Bestavros, "Statistical rate monotonic scheduling," in *19th Real-Time Systems Symp. (RTSS)*. IEEE, 1998, pp. 123–132.
[6] G. Bernat, A. Burns, and A. Liamosi, "Weakly hard real-time systems," *IEEE Trans. Computers*, vol. 50, no. 4, pp. 308–321, 2001.
[7] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed. Kluwer, 2011.

[8] C. Lin, T. Kaldewey, A. Povzner, and S. Brandt, "Diverse soft real-time processing in an integrated system," in *Real-Time Systems Symp. (RTSS)*. IEEE, 2006, pp. 369–378.

[9] S. Manolache, P. Eles, and Z. Peng, "Schedulability analysis of applications with stochastic task execution times," *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 4, p. 706–735, 2004.

[10] A. Friebe, A. V. Papadopoulos, and T. Nolte, "Identification and validation of markov models with continuous emission distributions for execution times," in *RTCSA*, 2020.

[11] R. I. Davis and L. Cucu-Grosjean, "A survey of probabilistic timing analysis techniques for real-time systems," *LITES: Leibniz Transactions on Embedded Systems*, pp. 1–60, 2019.

[12] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE Trans. Computers*, vol. 44, no. 12, pp. 1443–1451, 1995.

[13] P. Ramanathan, "Graceful degradation in real-time control applications using (m, k)-firm guarantee," in *Proc. IEEE 27th Int. Symp. on Fault Tolerant Computing*, 1997, pp. 132–141.

[14] R. West and C. Poellabauer, "Analysis of a window-constrained scheduler for real-time and best-effort packet streams," in *21st Real-Time Systems Symp. (RTSS)*. IEEE, 2000, pp. 239–248.

[15] L. Ahrendts, S. Quinton, and R. Ernst, "Finite ready queues as a mean for overload reduction in weakly-hard real-time systems," in *25th Real-Time Networks and Systems*. ACM, 2017, p. 88–97.

[16] H. Choi, H. Kim, and Q. Zhu, "Job-class-level fixed priority scheduling of weakly-hard real-time systems," in *RTAS*. IEEE, 2019, pp. 241–253.

[17] T. Yoshimoto and T. Ushio, "Optimal arbitration of control tasks by job skipping in cyber-physical systems," in *2011 IEEE/ACM 2nd Int. Conf.Cyber-Physical Systems*, 2011, pp. 55–64.

[18] S. Manolache, P. Eles, and Z. Peng, "Optimization of soft real-time systems with deadline miss ratio constraints," in *RTAS*, 2004.

[19] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *19th Real-Time Systems Symp.* IEEE, 1998, pp. 286–295.

[20] G. Buttazzo, M. Velasco, and P. Marti, "Quality-of-control management in overloaded real-time systems," *IEEE Trans. Computers*, vol. 56, no. 2, pp. 253–266, 2007.

[21] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. L. Bello, J. M. López, S. L. Min, and O. Mirabella, "Stochastic analysis of periodic real-time systems," in *23rd IEEE RTSS*. IEEE, 2002, pp. 289–300.

[22] A. Cervin, "Analysis of overrun strategies in periodic control tasks," *IFAC Proceedings Volumes*, vol. 38, no. 1, pp. 219–224, 2005.

[23] P. Pazzaglia, C. Mandrioli, M. Maggio, and A. Cervin, "DMAC: deadline-miss-aware control," *Dagstuhl Artifacts Ser.*, vol. 5, no. 1, pp. 03:1–03:3, 2019.

[24] W. Geelen, D. Antunes, J. P. M. Voeten, R. R. H. Schiffelers, and W. P. M. H. Heemels, "The impact of deadline misses on the control performance of high-end motion control systems," *IEEE Trans. Industrial Electronics*, vol. 63, no. 2, pp. 1218–1229, 2016.

[25] G. Koren and D. Shasha, "Skip-over: algorithms and complexity for overloaded systems that allow skips," in *RTSS*. IEEE, 1995, pp. 110–117.

[26] A. Marchand and M. Chetto, "Dynamic scheduling of periodic skippable tasks in an overloaded real-time system," in *AICCSA*, 2008, pp. 456–464.

[27] M. Harchol-Balter, "Open problems in queueing theory inspired by datacenter computing," *Queueing Systems*, vol. 97, no. 1, pp. 3–37, 2021.

[28] A. Chydzinski, "Queues with the dropping function and non-poisson arrivals," *IEEE Access*, vol. 8, pp. 39 819–39 829, 2020.

[29] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM T. on Network.*, vol. 1, no. 4, pp. 397–413, 1993.

[30] C.-W. Feng, L.-F. Huang, C. Xu, and Y.-C. Chang, "Congestion control scheme performance analysis based on nonlinear red," *IEEE Systems Journal*, vol. 11, no. 4, pp. 2247–2254, 2015.

[31] V. Rosolen, O. Bonaventure, and G. Leduc, "A RED discard strategy for ATM networks and its performance evaluation with TCP/IP traffic," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 3, pp. 23–43, 1999.

[32] C. Denninnart, J. Gentry, A. Mokhtari, and M. A. Salehi, "Efficient task pruning mechanism to improve robustness of heterogeneous computing systems," *J. Parallel Distributed Computing*, 2020.

[33] C. Denninnart, J. Gentry, and M. A. Salehi, "Improving robustness of heterogeneous serverless computing systems via probabilistic task pruning," in *IPDPS Workshops*. IEEE, 2019, pp. 6–15.

[34] J. Gentry, C. Denninnart, and M. A. Salehi, "Robust dynamic resource allocation via probabilistic task pruning in heterogeneous computing systems," in *IPDPS*. IEEE, 2019, pp. 375–384.

[35] A. Mokhtari, C. Denninnart, and M. A. Salehi, "Autonomous task dropping mechanism to achieve robustness in heterogeneous computing systems," *arXiv preprint arXiv:2005.11050*, 2020.

[36] M. A. Salehi, J. Smith, A. A. Maciejewski, H. J. Siegel, E. K. Chong, J. Apodaca, L. D. Briceno, T. Renner, V. Shestak, J. Ladd *et al.*, "Stochastic-based robust dynamic resource allocation for independent tasks in a heterogeneous computing system," *Journal of Parallel and Distributed Computing*, vol. 97, pp. 96–111, 2016.

[37] T. Patel, Z. Liu, R. Kettimuthu, P. Rich, W. Allcock, and D. Tiwari, "Job characteristics on large-scale systems: long-term analysis, quantification, and implications," in *SC'20*. IEEE, 2020.

[38] C. S. Yeo and R. Buyya, "A taxonomy of market-based resource management systems for utility-driven cluster computing," *Software: Practice and Experience*, vol. 36, no. 13, pp. 1381–1419, 2006.

[39] L.-C. Canon, A. K. W. Chang, Y. Robert, and F. Vivien, "Scheduling independent stochastic tasks under deadline and budget constraints," *The Int. J. of High Perf. Computing Applications*, vol. 34, no. 2, pp. 246–264, 2020.

[40] O. Häggström, *Finite Markov Chains and Algorithmic Applications*. Cambridge University Press, 2002.

[41] D. Feitelson, "Workload modeling for computer systems performance evaluation," *Version 1.0.3*, pp. 1–607, 2014.

BIOGRAPHIES

Yiqin Gao is a researcher in the Center of High Performance Computing at Shanghai Jiao Tong University. She obtained her PhD degree from ENS Lyon, France in 2021. She is mainly interested in scheduling techniques and resource allocation problems. See http://perso.ens-lyon.fr/yiqin.gao/ for further information.

Guillaume Pallez is a tenured researcher at Inria Bordeaux – Sud-Ouest. His research interests include algorithm design and scheduling techniques for parallel and distributed platforms (I/O-scheduling, data-aware scheduling, stochastic scheduling etc). He was a recipient of the 2019 IEEE TCHPC Early Career researcher award. See http://people.bordeaux.inria.fr/gaupy/ for further information.

Yves Robert is a Full Professor in the Computer Science Laboratory LIP at ENS Lyon. He is a Fellow of the IEEE and a Senior Member of Institut Universitaire de France. He has been awarded the 2014 IEEE TCSC Award for Excellence in Scalable Computing, the 2016 IEEE TCPP Outstanding Service Award, and the 2020 IEEE CS Charles Babbage Award. He holds a Visiting Scientist position at the Innovative Computing Laboratory at University of Tennessee, Knoxville, since 2011. His main research interests are scheduling techniques, parallel algorithms and resilient approaches for large-scale platforms. See http://graal.ens-lyon.fr/~yrobert/ for further information.

Frédéric Vivien is a Senior Research Scientist at INRIA. His main research interests are scheduling techniques, parallel algorithms, and resilient approaches for large-scale platforms. See http://perso.ens-lyon.fr/frederic.vivien/ for further information.