



HAL
open science

Scheduling Strategies for Overloaded Real-Time Systems

Yiqin Gao, Guillaume Pallez, Yves Robert, Frédéric Vivien

► **To cite this version:**

Yiqin Gao, Guillaume Pallez, Yves Robert, Frédéric Vivien. Scheduling Strategies for Overloaded Real-Time Systems. [Research Report] RR-9455, Inria - Research Centre Grenoble – Rhône-Alpes. 2022, pp.1-48. hal-03580853

HAL Id: hal-03580853

<https://inria.hal.science/hal-03580853>

Submitted on 18 Feb 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Scheduling Strategies for Overloaded Real-Time Systems

Yiqin Gao, Guillaume Pallez, Yves Robert, Frédéric Vivien

**RESEARCH
REPORT**

N° 9455

February 2022

Project-Teams ROMA, TADaaM

ISRN INRIA/RR--9455--FR+ENG

ISSN 0249-6399



Scheduling Strategies for Overloaded Real-Time Systems

Yiqin Gao^{*}, Guillaume Pallez[†], Yves Robert^{‡§}, Frédéric Vivien[‡]

Project-Teams ROMA, TADaaM

Research Report n° 9455 — February 2022 — 48 pages

Abstract: This paper introduces and assesses novel strategies to schedule firm real-time jobs on an overloaded server. The jobs are released periodically and have the same relative deadline. Job execution times obey an arbitrary probability distribution and can take unbounded values (no WCET). We introduce three control parameters to decide when to start or interrupt a job. We couple this dynamic scheduling with several admission policies and investigate several optimization criteria, the most prominent being the *Deadline Miss Ratio* (DMR). Then 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 14 different probability distributions; the results nicely demonstrate how the new control parameters help improve system performance compared with traditional approaches. In particular, we show that (i) the best admission policy is to admit all jobs; (ii) the key control parameter is to upper bound the start time of each job; (iii) the best scheduling strategy decreases the DMR by up to 0.35 over traditional competitors.

Key-words: real-time system, firm jobs, overloaded system, admission policy, job interruption, Markov model, scheduling strategy.

Note: A short version of this work has appeared as a 4-page Work-In-Progress paper at RTSS'2021 [15].

^{*} Shanghai Jiao Tong University, China

[†] Inria Bordeaux, France

[‡] LIP, École Normale Supérieure de Lyon, CNRS & Inria, France

[§] University of Tennessee Knoxville, USA

**RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES**

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

Stratégies d'ordonnancement pour un système en temps-réel surchargé

Résumé : Ce travail présente et évalue de nouvelles stratégies d'ordonnancement pour exécuter des tâches périodiques en temps réel sur une plate-forme surchargée. Les tâches arrivent périodiquement et ont le même délai relatif pour leur exécution. Les temps d'exécution des tâches obéissent à une distribution de probabilité arbitraire et peuvent prendre des valeurs illimitées (pas de WCET). Certaines tâches peuvent être interrompues à leur admission dans le système ou bien en cours d'exécution. Nous introduisons trois paramètres de contrôle pour décider quand démarrer ou interrompre une tâche. Nous associons cet ordonnancement dynamique à plusieurs politiques d'admission et étudions plusieurs critères d'optimisation, le plus important étant le *Deadline Miss Ratio* (DMR). Ensuite, nous dérivons un modèle de Markov et utilisons sa distribution stationnaire pour déterminer la meilleure valeur de chaque paramètre de contrôle. Enfin, nous conduisons de vastes simulations avec 14 distributions de probabilité différentes ; les résultats démontrent bien comment les nouveaux paramètres de contrôle contribuent à améliorer les performances du système par rapport aux approches traditionnelles. En particulier, nous montrons que (i) la meilleure politique d'admission est d'admettre toutes les tâches; (ii) le paramètre de contrôle clé est de limiter le temps de début de chaque tâche après son admission; (iii) la meilleure stratégie de planification diminue le DMR jusqu'à 0,35 par rapport aux concurrents traditionnels.

Mots-clés : système en temps réel, système surchargé, interruption, politique d'admission, modèle de Markov model, stratégie d'ordonnancement

1 Introduction

Overloaded systems have become the norm in scientific computing. With the advent of parallel computing for many societal advances, traditional computing systems are under a much higher load than was typically the case a few years ago. High-Performance Computing (HPC) systems run with a utilization of almost 100% and a job waiting time of days [26]. In order to cope with this increase in submissions, several approaches are considered. In Cloud computing, the typical approach is to have dynamic pricing, with prices increasing as the resources become more scarce [32]. In HPC, high waiting times are usually a deterrent with important negative consequences, such as limiting the number of very large jobs on HPC platforms [26].

In this work, we focus on real-time systems, but we borrow techniques from HPC or Cloud computing. We consider a periodic task composed of many jobs that are released periodically and that must complete execution before some (relative) deadline D . To cope with overloading, several approaches have been proposed [2]. For weakly-hard systems, some job deadlines can be missed. There are two sub-categories then: either a job that missed its deadline must still be completed (*soft* jobs), or it can be ignored (*firm* jobs). For soft jobs, it is acceptable to miss some deadlines occasionally, but all jobs must be completed eventually. This specification is impossible to meet for heavily loaded systems, e.g., with a short release period and long job execution times. On the contrary, for firm systems, there is no value to complete a job after its deadline. The objective is then to minimize the *Deadline Miss Ratio* (DMR), i.e., the fraction of jobs that fail to complete execution before their deadline.

The main focus of this work is to introduce and assess several new scheduling techniques for firm real-time systems, which have many important applications, including multimedia, satellite-based tracking, financial forecast, and robotic assembly lines [21, 23]. When not all jobs can meet their deadline, the scheduler can use a restrictive admission policy and decide to drop some jobs upon release. More importantly, the scheduler can also interrupt the job that is 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 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 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 before interruption d_{\max} (e.g., which can be smaller than the deadline D to favor next jobs); (ii) the maximal execution time l_{\max} allotted to the job (e.g., which can depend upon the probability distribution of job execu-

tion times); and (ii) the maximum time elapsed between release time and start-up time s_{\max} (e.g., which can be tuned according to system pressure).

Most real-time systems focus on jobs whose execution times obey a probability distribution with bounded support $[a, b]$, where $a \geq 0$. The value b is called the *Worst-Case Execution Time* (WCET) and the typical assumption is that $b \leq D$. Enforcing a WCET is mandatory for hard systems where deadlines cannot be missed. It is also very important for soft systems where all jobs must eventually complete. But for firm systems, one can relax the assumption and deal with standard distributions with infinite support (exponential, log-normal, half-normal, truncated normal, etc) chosen to encompass a wide range of characteristics.

Altogether, this work makes the following main contributions to the study of firm real-time systems :

- we deal with job execution times which obey an arbitrary probability distribution with unbounded support, hence, no WCET
- 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 3.2)
- 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 14 different probability distributions that lead to distinct system behaviors (see Sections 6 and 7)
- we show that (i) the best admission policy is to admit all jobs; (ii) the key control parameter is to upper bound the start 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. In the literature, real-time systems are classified as *hard* and *weakly hard* [2]. For hard real-time jobs, no deadline should be missed: it is mandatory that each job completes before its deadline. On the contrary, for weakly-hard jobs, some deadlines can be missed. There are two sub-categories then: either a job that missed its deadline must still be completed (*soft* jobs), or it can be ignored (*firm* jobs). For soft jobs, it is acceptable to miss some deadlines occasionally, but all jobs must be completed eventually. On the contrary for firm jobs, there is no value to complete a job after its deadline.

When a real-time system is overloaded, it is unavoidable that some deadlines are missed. For firm jobs, 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) [18, 28, 31, 1, 7]¹. For some applications, additional constraints, such as equally spacing the jobs matching their deadlines, may be enforced [33]. 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!

For soft jobs, the approach is different. All jobs have to be completed eventually. When the system is overloaded, an approach is either to increase the deadline, or to decrease the release period, or both [3, 4]. Another approach is to accumulate a backlog of jobs that are waiting for completion and to analyze the expected response time of the system [11]. In addition, some works 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 [6, 27],

¹In this work, we do not assume that jobs admit a finite worst-case execution time (WCET), which makes it impossible to guarantee any (m, k) constraint because some jobs can last for an arbitrary duration.

with the objective to minimize the deadline miss ratio (see Section 3 for a definition). Finally, a variant of the problem is dealt with in [16]: job i needs the output of job $i - 1$ to start, but can use that of job $i - 2$ if job $i - 1$ 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 [22, 24], and only skippable jobs (blue jobs in [22]) 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 [19] 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 [8], 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) [14] or other more complicated functions [13, 29]. 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 [9, 10, 17, 25, 30], 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.

3 Model and Optimization Criteria

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

3.1 Model

We consider a single periodic task whose jobs are periodically input to the system. The instance number i of the periodic task, called job 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 $d_i = r_i + D$, where τ is the period and D the relative deadline. If job i is not completed by its deadline d_i , it is considered as failed, its execution is interrupted, and any time spent to execute part of it has been wasted.

We assume that we have an *arbitrary deadline* $D > \tau$. (If $D \leq \tau$, the (trivial) solution is to never interrupt a job.) The problem becomes challenging when the system gets overloaded: when we have, say, $D = 5\tau$: 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. We consider a general framework and assume that job execution times obey some probability distributions with unbounded support $[0, \infty]$.

3.2 Optimization Criteria

We start with the most important criterion:

Deadline miss ratio (DMR) [11, 5]

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. Note that this metric is directly related to the *job execution rate* metric, which corresponds to the expected number of jobs executed per time unit.

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

Utilization [26]

This platform-oriented criterion corresponds to the proportion of time spent doing computation that ends up in 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 than more shorter ones.

Mean response time of successfully completed jobs [11, 26]

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 qualitative information about the various solutions.

Mean rejection time

This new criterion is important for an overloaded 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 i there are already m jobs in the queue, job i is rejected. Strategies using a queue of size m will be tagged by the prefix $\text{QUEUE}(m)$ 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 i -th job 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 D . Hence, a job released at time r will have its execution stopped at time $r + D$ 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 $r + D$, 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 D 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: $\tau \leq d_{\max} \leq D$.

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 starting 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 starting time of jobs. With such a bound, any job S 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 S 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 S 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: $0 \leq s_{\max} \leq d_{\max} - \tau$.

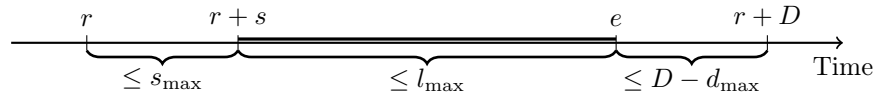


Figure 1: A successful job released at time r whose relative deadline is D , which starts its execution after s units of time, and that finishes at time e satisfies the constraints s_{\max} , l_{\max} , d_{\max} .

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, $(d_{\max}, l_{\max}, s_{\max}) = (D, D, D - \tau)$.

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 $(d_{\max}, l_{\max}, s_{\max}) = (D, D, m\tau - 1)$: 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 $d_{\max} = D$, $l_{\max} = d_{\max}$, $s_{\max} = d_{\max} - \tau$. 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 we are using $AP = AA$. The default objective is to minimize the DMR.

5 Markov Model

If the probability distribution \mathcal{D} 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 [11] 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{D} is initially continuous rather than discrete) and by setting notations. Then, in Section 5.2, we construct the Markov chain when all tasks 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 D both last an integral number of quanta. In other words, we have $\tau = \lfloor \frac{\tau}{q} \rfloor q$ and $D = \lfloor \frac{D}{q} \rfloor q$.

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 D are integers.

We approximate the probability distribution \mathcal{D} of job execution times via discretization. Let f be the probability density function of \mathcal{D} . Let p_l be the probability that a job execution time is equal to l (quanta) in the discretized version of \mathcal{D} . 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 S released at a time r . Let s be the time job S has to wait after its arrival in the system until the server is available. Then the server will be available for job S at the date $r + s$. 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 $s \leq l_{\max} + d_{\max} - \tau$. Altogether, this gives us $0 \leq s \leq \sigma$ with $\sigma = \min\{s_{\max} + l_{\max}, d_{\max}\} - \tau$. Job S is followed by a job T 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 T if it was available at time $r + s$ for job S .

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 S in the absence of any constraint (D , s_{\max} , l_{\max} , and d_{\max}). Without those constraints, the execution of job S 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 S finished before the release time of job T). 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 S is not launched by definition of s_{\max} . Therefore the server becomes available for job T at the time it became available for job S . Therefore, for any s and t such that $s_{\max} + 1 \leq s \leq \sigma$ and

$0 \leq t \leq \sigma$:

$$\begin{cases} \mathcal{P}_{s, \max\{0, s-\tau\}} = 1 \\ \mathcal{P}_{s,t} = 0 \end{cases} \quad \text{if } t \neq \max\{0, s-\tau\}.$$

- *Case $0 \leq s \leq s_{\max}$ and $t = 0$.*

The server is available at time $t = 0$ for job T if the execution of job S 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 two constraints:

$$l \leq l_{\max} \quad \text{and} \quad s + l \leq d_{\max}.$$

Hence, $l \leq \min\{l_{\max}, d_{\max} - s\}$. Then the job execution completes and leads to $t = 0$ if and only if: $1 \leq l \leq \min\{l_{\max}, d_{\max} - s, \tau - s\}$ (because in this case $l' = l$).

Moreover, the job execution is stopped and leads to $t = 0$ if and only if: $\min\{l_{\max}, d_{\max} - s\} + 1 \leq l$ and $l' \leq \tau - s$ with $l' = \min\{l_{\max}, d_{\max} - s\}$.

Altogether, we have:

$$\mathcal{P}_{s,0} = \sum_{l=1}^{\min\{l_{\max}, d_{\max}-s, \tau-s\}} pl + \sum_{\substack{l \geq \min\{l_{\max}, d_{\max}-s\}+1 \\ \min\{l_{\max}, d_{\max}-s\} \leq \tau-s}} pl$$

Remark that the second sum is null except in the degenerate case where the only existing state is $s = 0$ ($l_{\max} = \tau$ or $d_{\max} = \tau$). Then we obviously have $\mathcal{P}_{0,0} = 1$.

- *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:

$$l \leq l_{\max} \quad \text{and} \quad s + l \leq d_{\max}.$$

Hence, $l \leq \min\{l_{\max}, d_{\max} - s\}$ which is equivalent to $t \leq s - \tau + \min\{l_{\max}, d_{\max} - s\}$. The case $t = s - \tau + \min\{l_{\max}, d_{\max} - s\}$ contains both situations where the job execution completes and situations where the job execution is stopped. Hence, the following decomposition into subcases:

- *Case* $1 \leq t \leq s - \tau + \min\{l_{\max}, d_{\max} - s\} - 1$. 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 $t = s - \tau + \min\{l_{\max}, d_{\max} - s\}$. This corresponds to all remaining possible cases. Hence,

$$\mathcal{P}_{s,t} = 1 - \sum_{l=1}^{\min\{l_{\max}, d_{\max} - s\} - 1} p_l.$$

- *Case* $t > s - \tau + \min\{l_{\max}, d_{\max} - s\}$. None of these cases can happen. Hence,

$$\mathcal{P}_{s,t} = 0.$$

5.3 Other Admission Policies

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

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

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 $\mathcal{P}_{(i-1)\sigma+s,t}^{per}$, with $1 \leq i \leq \tau_A$, $0 \leq s, t \leq \sigma$, denote the probability that, if the server was available at time s for job S and if job S was the n -th job released in the system such that $1 + (n \bmod \tau_A) = i$, then the server will be available at time t for job T . Then we have:

- Case $\mathcal{A}[i] = \text{TRUE}$ (job S is admitted).

$$\mathcal{P}_{(i-1)\sigma+s,t}^{per} = \mathcal{P}_{s,t}.$$

- Case $\mathcal{A}[i] = \text{FALSE}$.

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

Queues of size m . When all jobs were admitted, we were able to fully characterize the system for job S 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 δ (determined later) describing the state of the queue. For $0 \leq l \leq \delta$, $\vec{Q}[l]$ is the number of jobs released *prior* to S 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 $\{0, \dots, m + 1\}$. Note that, because the execution of S starts at time $r + s$, the system does not contain any jobs released prior to S 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, $\delta \leq \lfloor \frac{\sigma}{\tau} \rfloor$.

We need to compute the probability $\mathcal{P}_{s, \vec{Q}, t, \vec{R}}^{queue}$ that the server is available at time $(r + \tau) + t$ for job T with a queue state \vec{R} if it was available at time $r + s$ for job S with queue state \vec{Q} . We start by characterizing the queue state if the server is available at time $r + t$ for job T . 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 S is not admitted. The only jobs released prior to T that may be present in the system are jobs released prior to S . 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 T is one period (of length τ) later than for job S .

- *Case $\vec{Q}[0] < m + 1$.* Then job S 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 S is admitted into the system, the probability that the server is available at time $r + t$ for job T 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 S is not admitted, then the server is available for job T at time $r + s$. Altogether, we obtain:

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

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 [20].

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.*

Proof. We prove that all Markov chains are irreducible and aperiodic. In the following, we restrict to the case when all jobs are admitted (AA policy). The proof can then be easily extended to the other admission policies.

If $l_{\max} = \tau$ or $d_{\max} = \tau$, the execution of any job will be completed or interrupted before the next job is released. The system is then trivial, the Markov chain has a single state, and the theorem is obviously true. Hence, in the following we assume that we have $l_{\max} > \tau$ and $d_{\max} > \tau$. In the proof, we assume that all possible execution times for a job do actually happen. That is, for any duration $l \in \{1 \dots D\}$, $p_l > 0$. We first show that all states are reachable. By definition of σ , the range of (theoretically) possible states is $[0; \sigma]$. Let us “forget” about d_{\max} for a short while. The server is available for job S at time $r + s$. Because $l_{\max} > \tau$ and $p_{\tau+1} > 0$, with probability $p_{\tau+1}$ the execution of job S will last for a time $\tau + 1$ and the server will be available for job T at time $r + s + \tau + 1 = (r + \tau) + s + 1$. Therefore, if the state of the system is s for job S , it will be in the state $t = s + 1$ for job T with probability $p_{\tau+1} > 0$. Now, taking d_{\max} into account, this is true as long as $s \leq s_{\max}$ and $s + \tau + 1 \leq d_{\max}$.

Let us first consider the case $s + \tau + 1 = d_{\max}$. If we reached it, this means that $s_{\max} + l_{\max} \geq d_{\max}$, because $s_{\max} \geq s$ and $l_{\max} \geq \tau + 1$. Therefore, the possible range of values is $[0; d_{\max} - \tau]$, and we have established that all these states are reachable. Now consider the other case. Namely, we reached the case $s = s_{\max}$ without, in the process, having jobs interrupted because of the value of d_{\max} . Then, with probability $p_{\tau+j} > 0$, with $j \in [1; \min\{l_{\max}; d_{\max} - s_{\max}\} - \tau]$, the execution of job S lasts until the time $r + s_{\max} + \tau + j = (r + \tau) + s_{\max} + j$. Hence, all states in the range $[s_{\max} + 1; \min\{s_{\max} + l_{\max}; d_{\max}\} - \tau]$ are reachable. Therefore, all possible states are reachable from the initial state, the state $s = 0$.

Now consider any state $s > 0$. With probability $p_1 > 0$, the next state is $t = \max\{0, s + 1 - \tau\}$. If $\tau > 1$, $t < s$, and from any state a strictly “smaller” state is reachable. Therefore, there is a path from any state to the state 0. This shows that the Markov chain is irreducible. Furthermore, there is a loop from state 0 to itself. Indeed, with probability $\sum_{k=1}^{\tau} p_k > 0$ a job execution takes no longer than the period τ . Hence, if the state before the execution of the job was 0, this is also the case after its execution. Hence, the period of state $s = 0$ is one. And because the chain is irreducible, all states have

period one too. Hence, the Markov chain is aperiodic when $\tau > 1$.

If $\tau = 1$, there is a loop from state any state s to itself. Let $l \geq 1$ be the length of the execution of job S before it either completes or is killed. Then $t = \max\{0, s + l - \tau\} = \max\{0, s + l - 1\} = s + l - 1 \geq s$. Furthermore, if $s \leq s_{\max}$, there is a loop from state s to itself because $p_1 > 0$. Hence, the period of any state $s \leq s_{\max}$ is one. If $s > s_{\max}$, there is a path from state s to state $s - 1$, and then to state s_{\max} . Therefore, there is a path from any state to the state s_{\max} . This shows that the Markov chain is irreducible. Therefore, all states have period one too. Hence the Markov chain is aperiodic when $\tau = 1$. This concludes the proof. \square

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, but only for DMR (deadline miss ratio) for the other admission policies (the other criteria can be derived similarly).

- **Deadline miss ratio.** The probability of success is computed as

$$\mathbb{P}_{succ}^{AA} = \sum_{s=0}^{\sigma} \pi_{\infty}(s) \sum_{l=1}^{\min\{l_{\max}, d_{\max}-s\}} p_l$$

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

Under a random admission policy of parameter α , jobs are rejected with probability $1 - \alpha$. Hence:

$$\mathbb{P}_{succ}^{rand} = \alpha \sum_{s=0}^{\sigma} \pi_{\infty}(s) \sum_{l=1}^{\min\{l_{\max}, d_{\max}-s\}} p_l$$

When admissions are regulated through a pattern, only jobs released at ranks corresponding to admissions can meet their deadlines. Hence:

$$\mathbb{P}_{succ}^{per} = \sum_{\substack{1 \leq i \leq \tau_A \\ A[i]=\text{TRUE}}} \sum_{s=0}^{\sigma} \pi_{\infty}(s, i) \sum_{l=1}^{\min\{l_{\max}, d_{\max}-s\}} p_l$$

Similarly, when admissions are regulated by queues, only jobs released when the queue is not full can meet their deadlines. Hence:

$$\mathbb{P}_{succ}^{queue} = \sum_{\substack{0 \leq s \leq \sigma \\ \forall \vec{Q} \text{ s.t. } \vec{Q}(0) < m}} \pi_{\infty}(s, \vec{Q}) \sum_{l=1}^{\min\{l_{\max}, d_{\max} - s\}} p_l$$

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

$$\mathbb{E}_{succ}^{AA} = \sum_{s=0}^{\sigma} \pi_{\infty}(s) \sum_{l=1}^{\min\{l_{\max}, d_{\max} - s\}} p_l$$

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

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

$$\sum_{s=0}^{\sigma} \pi_{\infty}(s) \sum_{l=1}^{\min\{l_{\max}, d_{\max} - s\}} p_l (s + l)$$

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

$$\mathbb{E}_1^{AA} = \sum_{s=0}^{\sigma} \pi_{\infty}(s) \left(\sum_{l \geq 1 + \min\{l_{\max}, d_{\max} - s\}} p_l \right) \times \min\{l_{\max}, d_{\max} - s\}$$

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

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

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

5.6 Complexity

We establish here the complexity of using the Markov chains to compute the optimal values of d_{\max} , l_{\max} and s_{\max} .

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

Proof. For a given admission policy and a choice of values for the parameters d_{\max} , l_{\max} , and s_{\max} we build the corresponding Markov chain following Sections 5.2 and 5.3. We obtain a Markov chain with β states. According to Theorem 1, this Markov chain is irreducible and aperiodic. We compute its stationary distribution in $O(\beta^3)$ steps. Using the stationary distribution and the objective function of choice, we can compute the performance of the admission policy and the choice of values of d_{\max} , l_{\max} , and s_{\max} . Then, the best value of d_{\max} , l_{\max} and/or s_{\max} is computed by trying all values; there are at most $\frac{D}{q}$ possible values for each of these three parameters. \square

We have $\beta \leq \frac{D-\tau}{q}$ when all jobs are admitted or when the admission is random, $\beta \leq \tau_{\mathcal{A}} \frac{D-\tau}{q}$ when jobs are admitted following a pattern of size $\tau_{\mathcal{A}}$ and $\beta \leq \frac{D-\tau}{q} (1+m)^\delta = \frac{D-\tau}{q} (1+m) \frac{D}{\tau}$ when jobs are admitted through a queue of size m .

For instance, with the AA policy, the complexity becomes $O(\frac{D}{q})^6$ 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 $O((\frac{D}{q})^3 \log(\frac{D}{q}))$.

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.

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

We use 8 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 [12]. 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 to illustrate different potential behaviors. Altogether, we have 14 distributions. To enable a direct comparison, we choose their parameters so that they all achieve an expectation $E(\mathcal{D}) = 1$.

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	$k \approx 0.411, \lambda = \frac{1}{\Gamma(1+\frac{1}{k})} \approx 0.324$ $k = 1.5, \lambda = \frac{1}{\Gamma(1+\frac{1}{k})} \approx 1.108$

Reduced evaluation set

In the experiments, we observed that many distributions had similar behaviors. Similarly, the behavior was quite consistent when the ratio $N = D/\tau$ 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 four distributions (two per generic behavior), those of the first column of Table 2 which categorizes the 14 distributions. Complete results for all distributions and values of N are available in the Appendix.

Table 2: Categorizing the behavior of the 14 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

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 : $s \mapsto (AA, (d_{\max}, l_{\max}, s_{\max}) = (D, D, s))$ (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 $[0, D]$. We consider four variants of it: S-MAX(xD) for $x \in \{0.25, 0.5, 0.75, 1\}$ to cover the whole scope of values.

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

- We repeat all experiments proposed in Section 6.1 ($14 \times 19 \times 5 = 1330$ experiments), 100 times.
- We measure the standard deviation of all scenarios (1330 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 13 of the Appendix).

From Figure 2, we see that the standard deviation with 10^6 jobs is lower than 0.001. This indicates that we can rely with high confidence on an

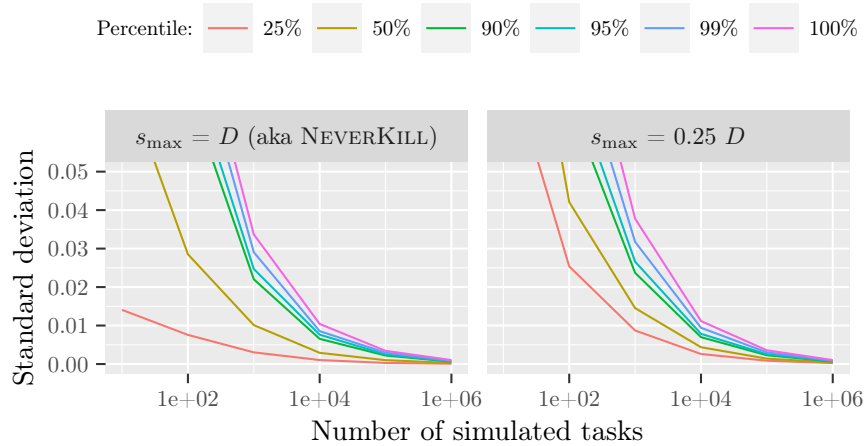


Figure 2: Evolution of the standard deviation of a set of 100 simulations for different s_{\max} heuristics, as a function of the number of simulated tasks simulated. For each number of tasks and each value of s_{\max} , we report percentiles on the standard deviation of the DMR for different periods,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

experimental evaluation with $\mathcal{N} = 10^6$ events. The time to execute a single evaluation with 10^6 tasks 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 quantum 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 $\text{BESTSMAX}(q)$ this algorithm. We study the performance of $q \mapsto \text{BESTSMAX}(q)$ when $q \in [0.001, 0.1]$ for all scenarios detailed in Section 6.1. Then, in Figure 3, we plot the percentage of deviation from the most precise model:

$$q \mapsto \left| 1 - \frac{\text{DMR}(\text{BESTSMAX}(q))}{\text{DMR}(\text{BESTSMAX}(0.001))} \right|$$

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

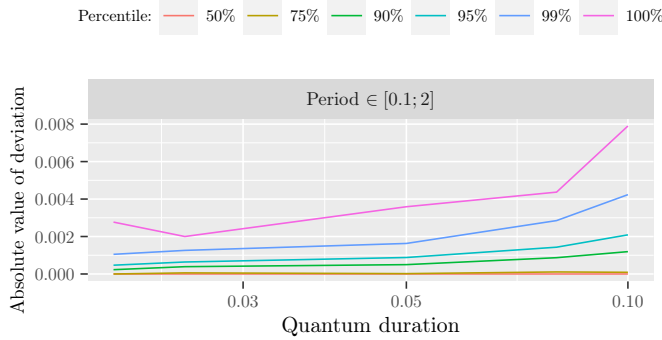


Figure 3: Absolute value of deviation from the performance simulated with 10^6 tasks predicted for heuristic S-MAX whose parameter s_{\max} was defined by a Markov chain using a quantum $q = 0.1$. For each quantum size, many different periods, deadlines, and distributions are evaluated whose percentiles (and worst-case) are presented.

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 $q = 0.001$ are always smaller than 0.01. 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 3.2).

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.

The most complete heuristic is the BESTDMAXLMAXSMAX heuristic which chooses the combination minimizing the DMR for a given period τ , a given distribution \mathcal{D} , and a given relative deadline D , 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

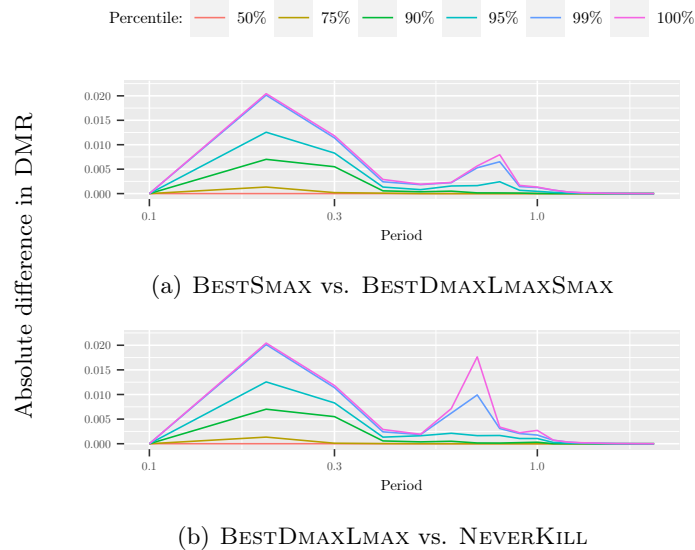


Figure 4: Absolute difference in the simulated performance between the named heuristics. For each value of the period τ , the main percentiles (and worst-case) are presented.

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 lead to a decrease of 0.02 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 constrained, with the strategy where none of the parameters are constrained, 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 $s_{\max} = D - \tau$ (not constraining).

Figure 5 displays the DMR of BESTSMAX and NEVERKILL while varying the period τ , the relative deadline D , and the distribution of job execution times \mathcal{D} . 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:

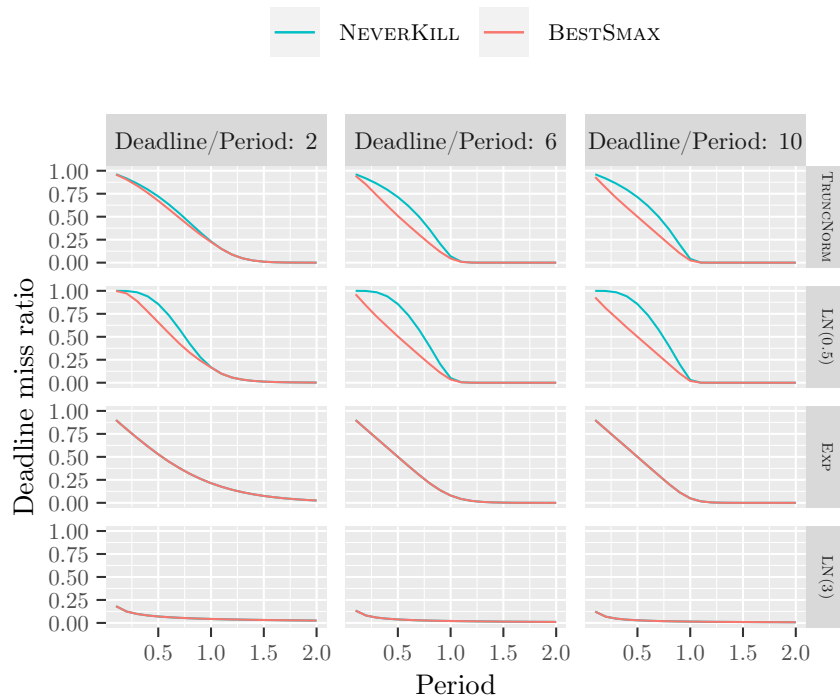


Figure 5: Absolute performance of job admission heuristics BESTSMAX and NEVERKILL (lower is better).

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)). 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 (D) with respect to the period (τ). On the contrary, the improvement of BESTSMAX with respect to NEVERKILL increases with the ratio $\frac{D}{\tau}$ and appears to reach a plateau when $\frac{D}{\tau} = 4$ 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 $D - \tau$. In

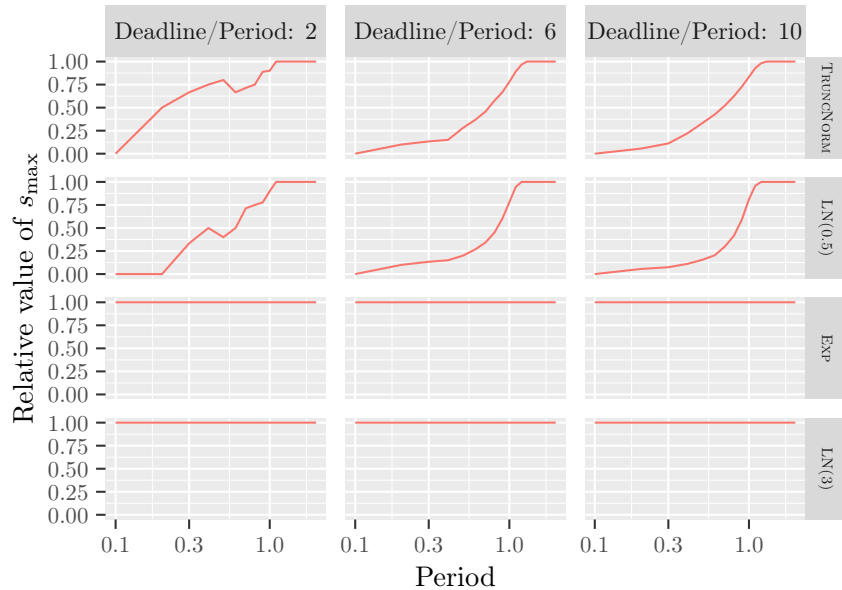


Figure 6: Best value for the s_{\max} threshold for job admission heuristic BESTSMAX expressed as a fraction of the maximum meaningful value $D - \tau$, when the quantum length is 0.1 time unit.

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.

One limitation of the BESTSMAX heuristic is its computational complexity. A remedy would be to be able to provide efficient heuristic 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 $(d_{\max}, l_{\max}, s_{\max}) = (D, D, m\tau - 1)$. 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 (overloaded 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.

Still, is it necessary to test all the $1 + \frac{D\tau}{q}$ possible values of s_{\max} to define BESTSMAX. We conjecture that the DMR of the function S-MAX : $s \mapsto (AA, (d_{\max}, l_{\max}, s_{\max}) = (D, D, s))$ 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.

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

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

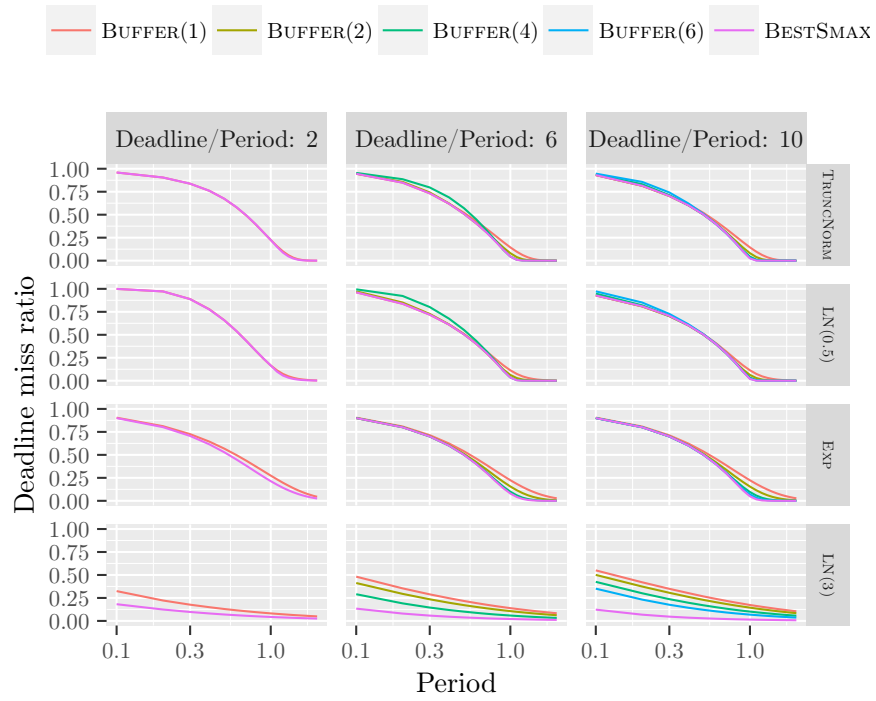


Figure 7: Deadline miss ratio of job admission heuristics BESTSMAX and BUFFER(m) for different period τ and distributions while varying q .

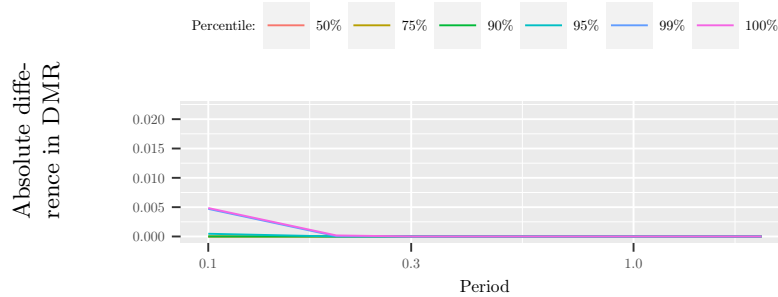


Figure 8: Absolute difference in the simulated performance between BESTSMAX vs. BINSMAX. For each value of the period τ , the main percentiles (and worst-case) are presented.

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.

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 D/τ). Whatever the distribution and set of parameters, *QUEUE*(m) never achieves better performance than *AA*.

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

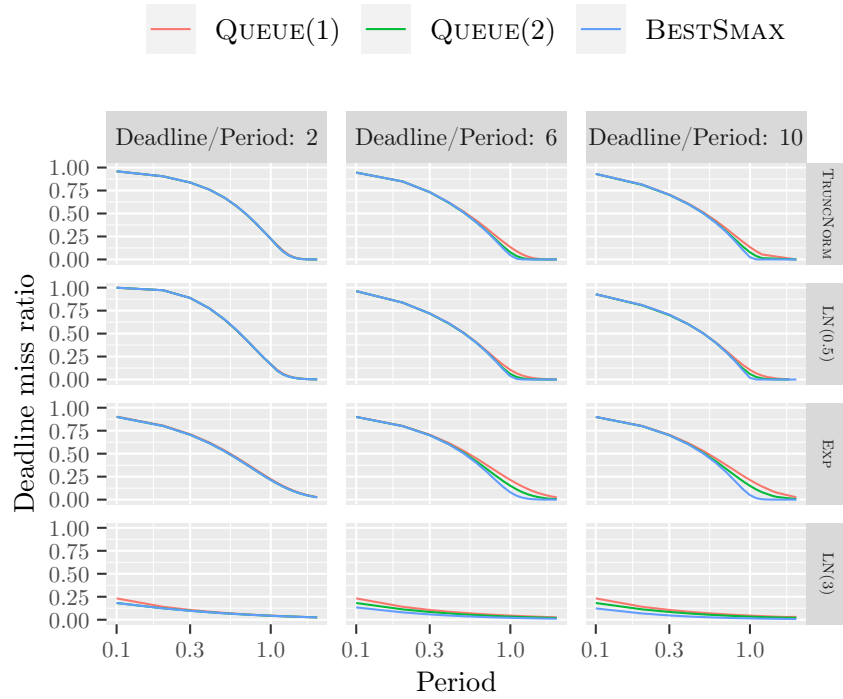


Figure 9: Deadline miss ratio of job admission policy $\text{QUEUE}(m)$ applied on BESTDMAXLMAXSMAX for different period τ and distributions.

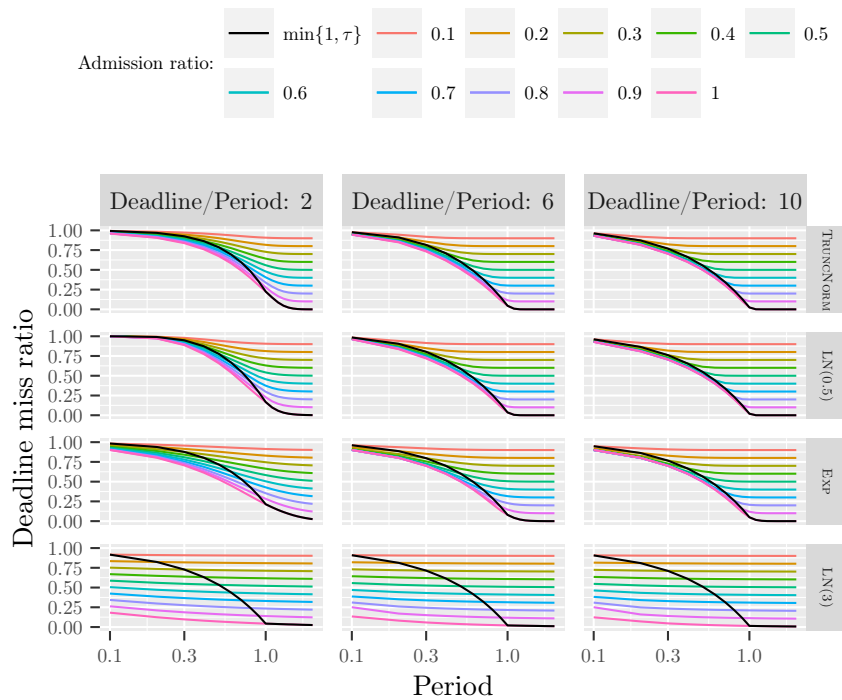


Figure 10: Deadline miss ratio of random job admission policy applied on BESTD MAXL MAXS MAX for different period τ and distributions.

does not perform well for any distribution.

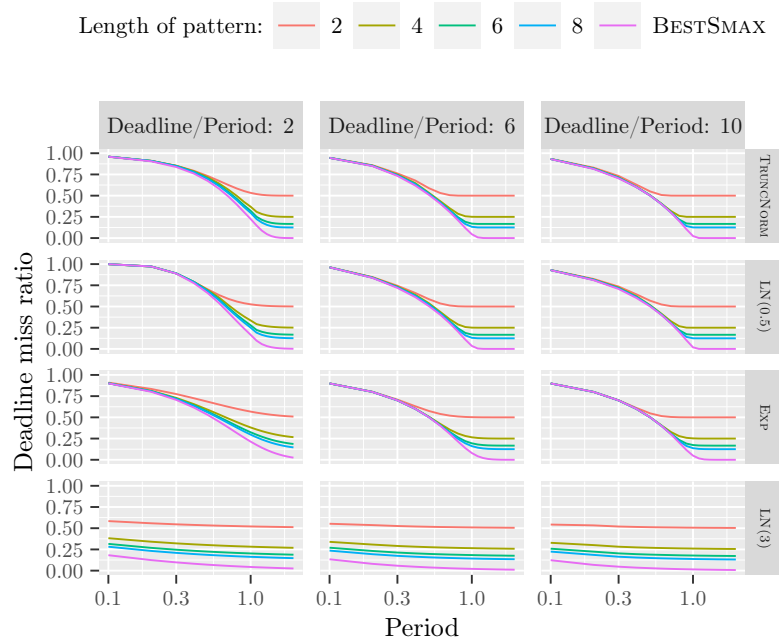


Figure 11: Deadline miss ratio of periodic job admission policy applied on BESTDMAXLMAXSMAX for different period τ and distributions.

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 3.2) to see if it may be interesting from another perspective to have different admission policies.

In Figure 7.3, we report the performance for the four criteria, for three of the admission policies. We report the performance of BESTDMAXLMAXS-MAX (for DMR minimization) either with a queue of size 1 (QUEUE(1)) or

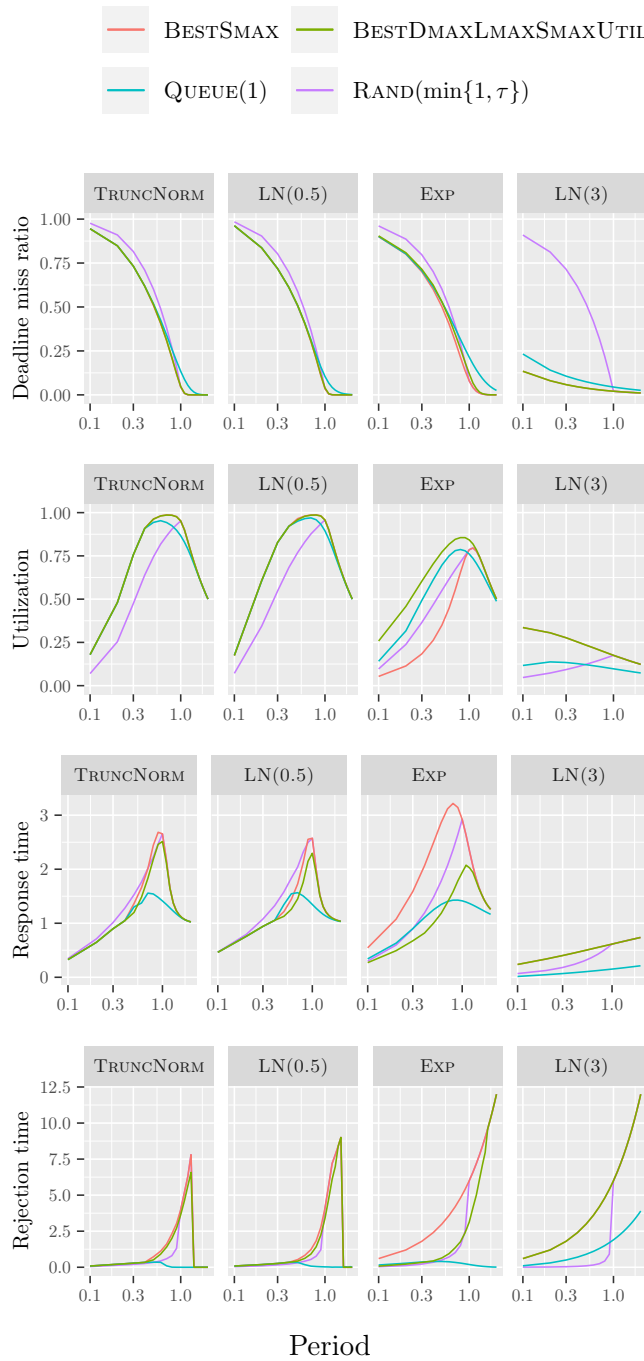


Figure 12: Performance of different optimization criteria for different heuristics, while varying period τ and fixing $N = 4$.

with a random admission policy with an admission ratio equal to the period ($\text{RAND}(\min\{1, \tau\})$ (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 $\text{RAND}(\min\{1, \tau\})$, 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. $\text{RAND}(\min\{1, \tau\})$ has an intermediate behavior due to a higher acceptance rate.

7.4 Summary

Our thorough evaluation provides several important insights about scheduling strategies for overloaded 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 trades-off 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 real-time jobs on an overloaded server. We have introduced three control parameters to decide when to start or interrupt a job. Coupled with several admission policies and 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 14 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.

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.

Finally, some applications may benefit from not treating all jobs equally; in a general setting, some particular jobs may have higher value for the user and, hence, should be executed successfully whenever possible. We believe that our approach can be extended to deal with such complicated scenarios.

References

- [1] L. Ahrendts, S. Quinton, and R. Ernst. Finite ready queues as a mean for overload reduction in weakly-hard real-time systems. In *Proc. 25th International Conference on Real-Time Networks and Systems (RTNS)*, page 88–97. ACM, 2017.
- [2] G. Bernat, A. Burns, and A. Liamosi. Weakly hard real-time systems. *IEEE Trans. Computers*, 50(4):308–321, 2001.
- [3] G. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *Proc. 19th IEEE Real-Time Systems Symposium*, pages 286–295, 1998.
- [4] G. Buttazzo, M. Velasco, and P. Marti. Quality-of-control management in overloaded real-time systems. *IEEE Trans. Computers*, 56(2):253–266, 2007.

-
- [5] 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*, 34(2):246–264, 2020.
 - [6] A. Cervin. Analysis of overrun strategies in periodic control tasks. *IFAC Proceedings Volumes*, 38(1):219–224, 2005.
 - [7] H. Choi, H. Kim, and Q. Zhu. Job-class-level fixed priority scheduling of weakly-hard real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 241–253, 2019.
 - [8] A. Chydzinski. Queues with the dropping function and non-poisson arrivals. *IEEE Access*, 8:39819–39829, 2020.
 - [9] C. Denninnart, J. Gentry, A. Mokhtari, and M. A. Salehi. Efficient task pruning mechanism to improve robustness of heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 2020.
 - [10] C. Denninnart, J. Gentry, and M. A. Salehi. Improving robustness of heterogeneous serverless computing systems via probabilistic task pruning. In *2019 IEEE IPDPS Workshops*, pages 6–15. IEEE, 2019.
 - [11] 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*, pages 289–300. IEEE, 2002.
 - [12] D. Feitelson. Workload modeling for computer systems performance evaluation. *Version 1.0.3*, pages 1–607, 2014.
 - [13] C.-W. Feng, L.-F. Huang, C. Xu, and Y.-C. Chang. Congestion control scheme performance analysis based on nonlinear red. *IEEE Systems Journal*, 11(4):2247–2254, 2015.
 - [14] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM T. on Network.*, 1(4):397–413, 1993.
 - [15] 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 Symposium (RTSS)*. IEEE Computer Society Press, 2021.
 - [16] 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*, 63(2):1218–1229, 2016.
 - [17] J. Gentry, C. Denninnart, and M. A. Salehi. Robust dynamic resource allocation via probabilistic task pruning in heterogeneous computing systems. In *IPDPS*, pages 375–384. IEEE, 2019.

-
- [18] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m, k) -firm deadlines. *IEEE Trans. Computers*, 44(12):1443–1451, 1995.
- [19] M. Harchol-Balter. Open problems in queueing theory inspired by data-center computing. *Queueing Systems*, 97(1):3–37, 2021.
- [20] O. Häggström. *Finite Markov Chains and Algorithmic Applications*. Cambridge University Press, 2002.
- [21] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer, USA, 2nd edition, 2011.
- [22] G. Koren and D. Shasha. Skip-over: algorithms and complexity for overloaded systems that allow skips. In *RTSS*, pages 110–117, 1995.
- [23] C. Lin, T. Kaldewey, A. Povzner, and S. Brandt. Diverse soft real-time processing in an integrated system. In *RTSS*, pages 369–378, 2006.
- [24] A. Marchand and M. Chetto. Dynamic scheduling of periodic skippable tasks in an overloaded real-time system. In *AICCSA*, pages 456–464, 2008.
- [25] 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.
- [26] 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*, pages 1–17. IEEE, 2020.
- [27] P. Pazzaglia, C. Mandrioli, M. Maggio, and A. Cervin. DMAC: deadline-miss-aware control. *Dagstuhl Artifacts Ser.*, 5(1):03:1–03:3, 2019.
- [28] P. Ramanathan. Graceful degradation in real-time control applications using (m, k) -firm guarantee. In *Proc. IEEE 27th Int. Symp. on Fault Tolerant Computing*, pages 132–141, 1997.
- [29] 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*, 29(3):23–43, 1999.
- [30] 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*, 97:96–111, 2016.

- [31] R. West and C. Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort packet streams. In *Proc. 21st IEEE Real-Time Systems Symposium*, pages 239–248, 2000.
- [32] C. S. Yeo and R. Buyya. A taxonomy of market-based resource management systems for utility-driven cluster computing. *Software: Practice and Experience*, 36(13):1381–1419, 2006.
- [33] 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*, pages 55–64, 2011.

Appendix

This section includes all figures for the evaluation, with the 14 probability distributions.

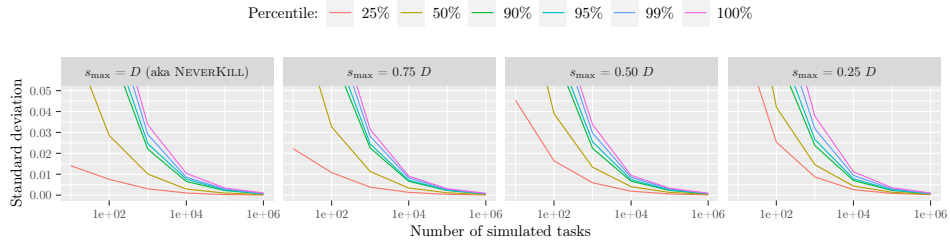


Figure 13: Evolution of the standard deviation of a set of 100 simulations for different s_{\max} heuristics, as a function of the number of simulated tasks simulated. For each number of tasks and each value of s_{\max} , we report percentiles on the standard deviation of the DMR for different periods, deadlines, and distributions.

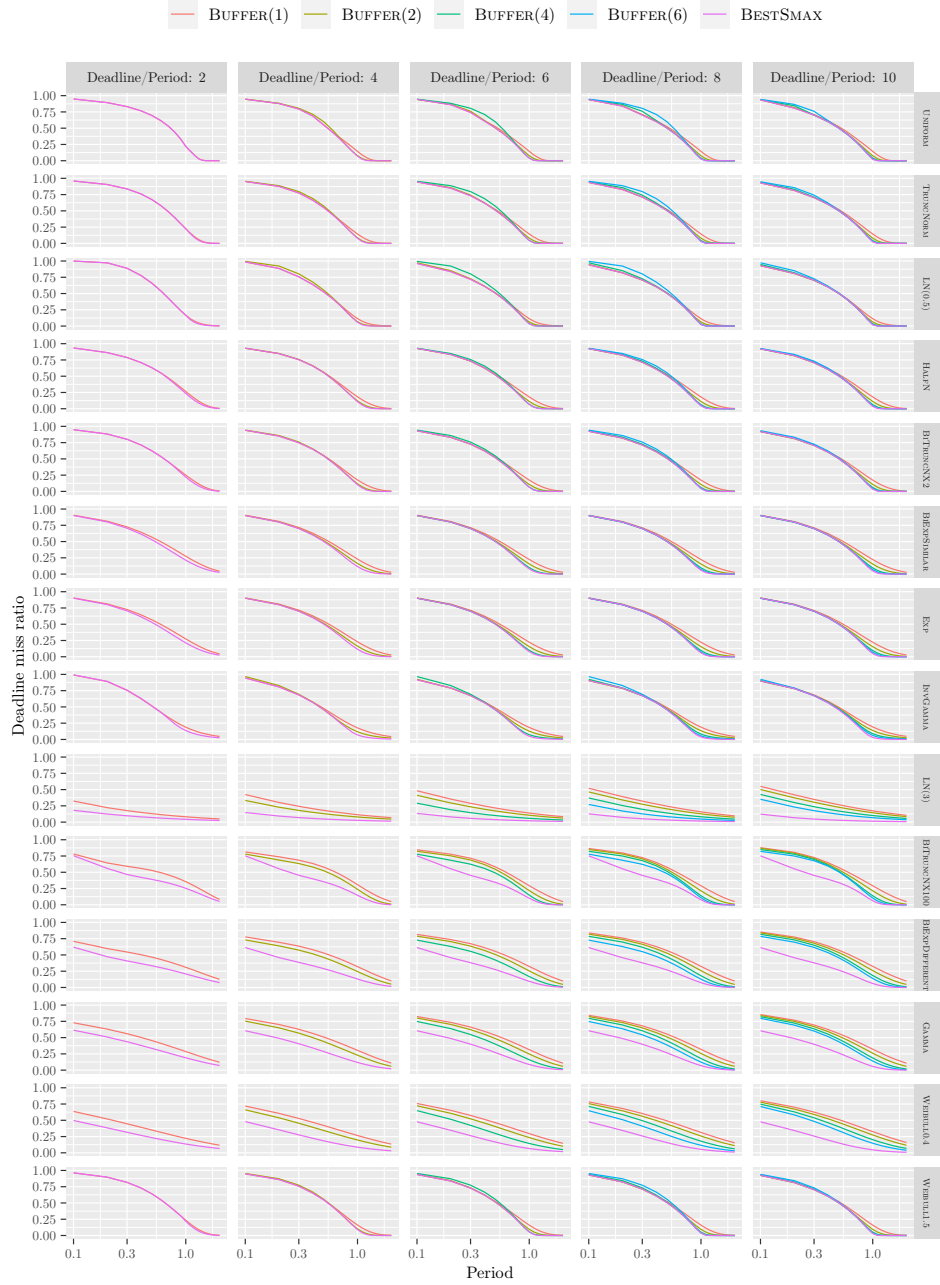


Figure 14: Deadline miss ratio of job admission heuristics BESTSMAX and $BUFFER(m)$ for different period τ and distributions while varying q , the performance are assessed through simulations of 10^6 jobs.

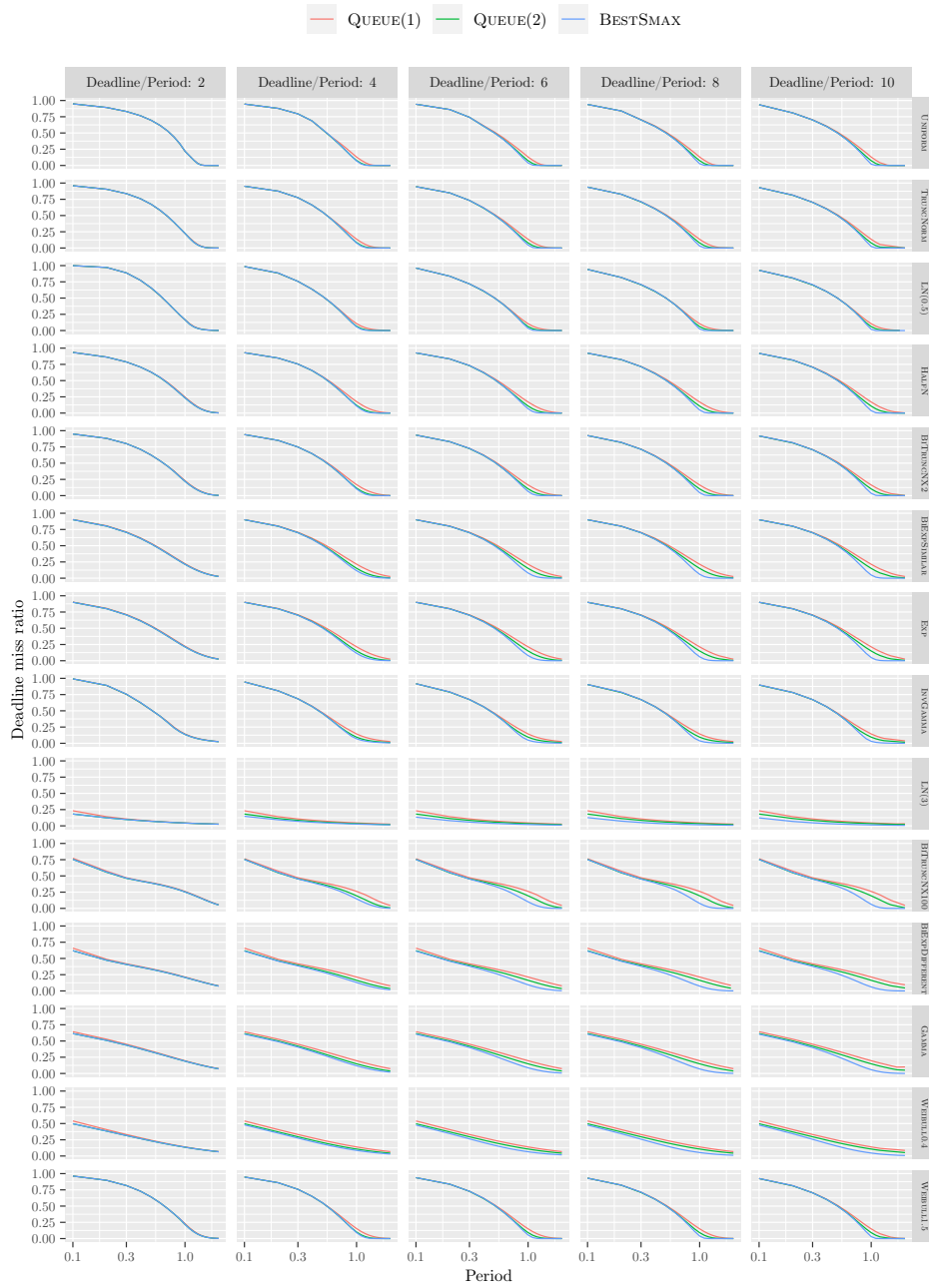


Figure 15: Deadline miss ratio of job admission policy $QUEUE(m)$ applied on $BESTDMAXLMAXSMAX$ for different period τ and distributions, the performance are assessed through simulations of 10^6 jobs.

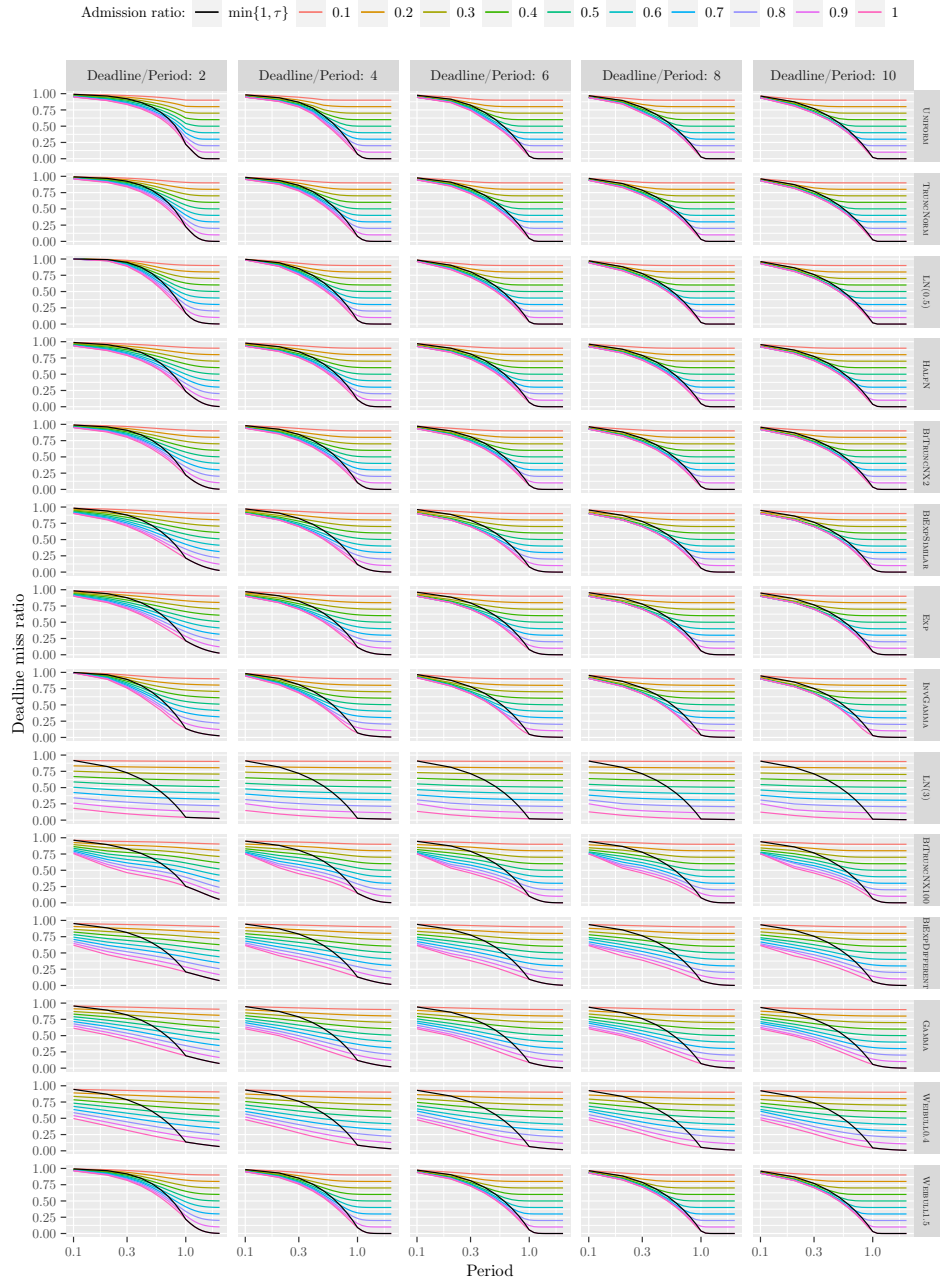


Figure 16: Deadline miss ratio of random job admission policy applied on BESTDMAXLMAXSMAX for different period τ and distributions, the performance are assessed through simulations of 10^6 jobs.

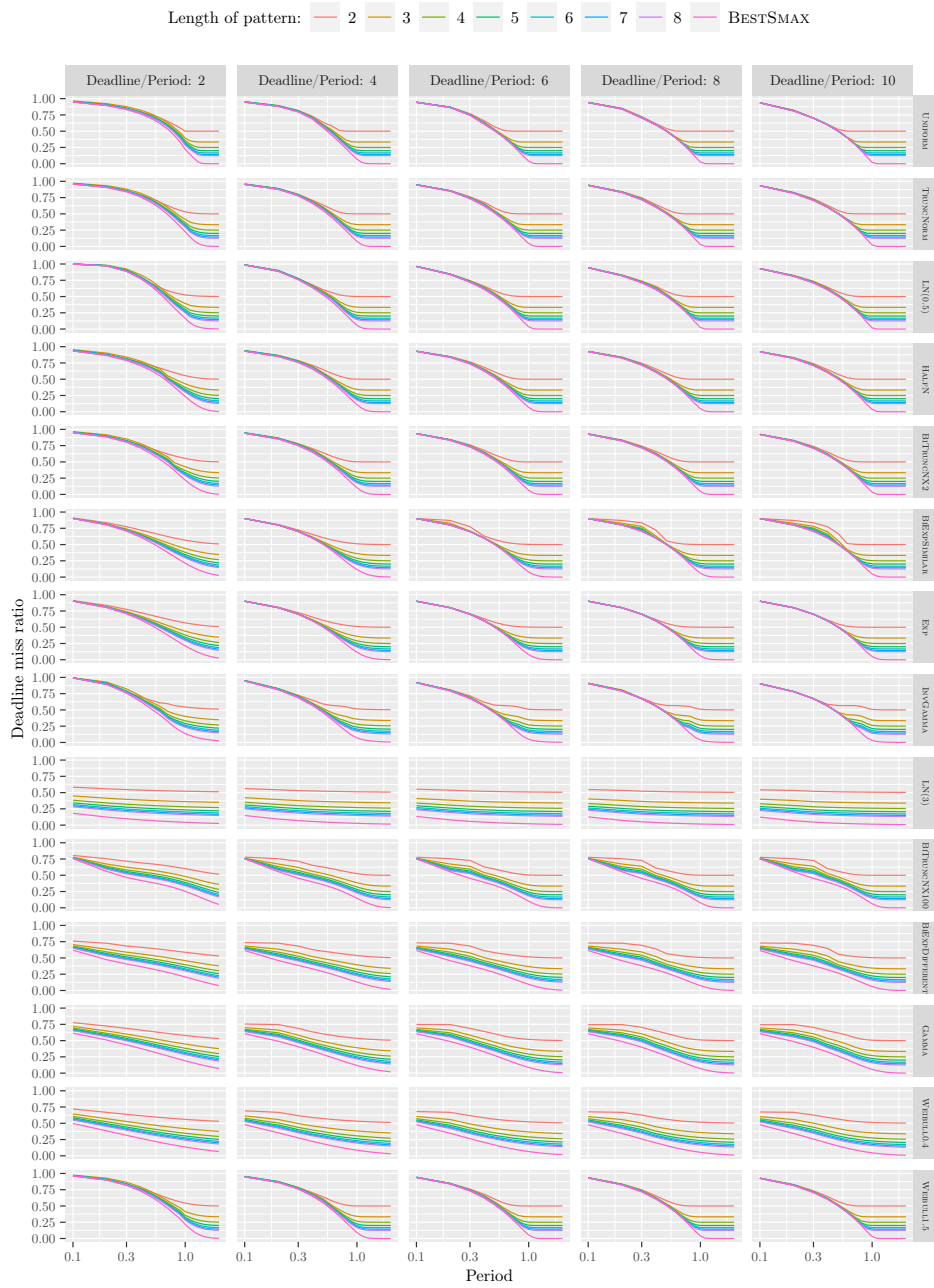


Figure 17: Deadline miss ratio of periodic job admission policy applied on BESTDMLMAXSMAX for different period τ and distributions, the performance are assessed through simulations of 10^6 jobs.

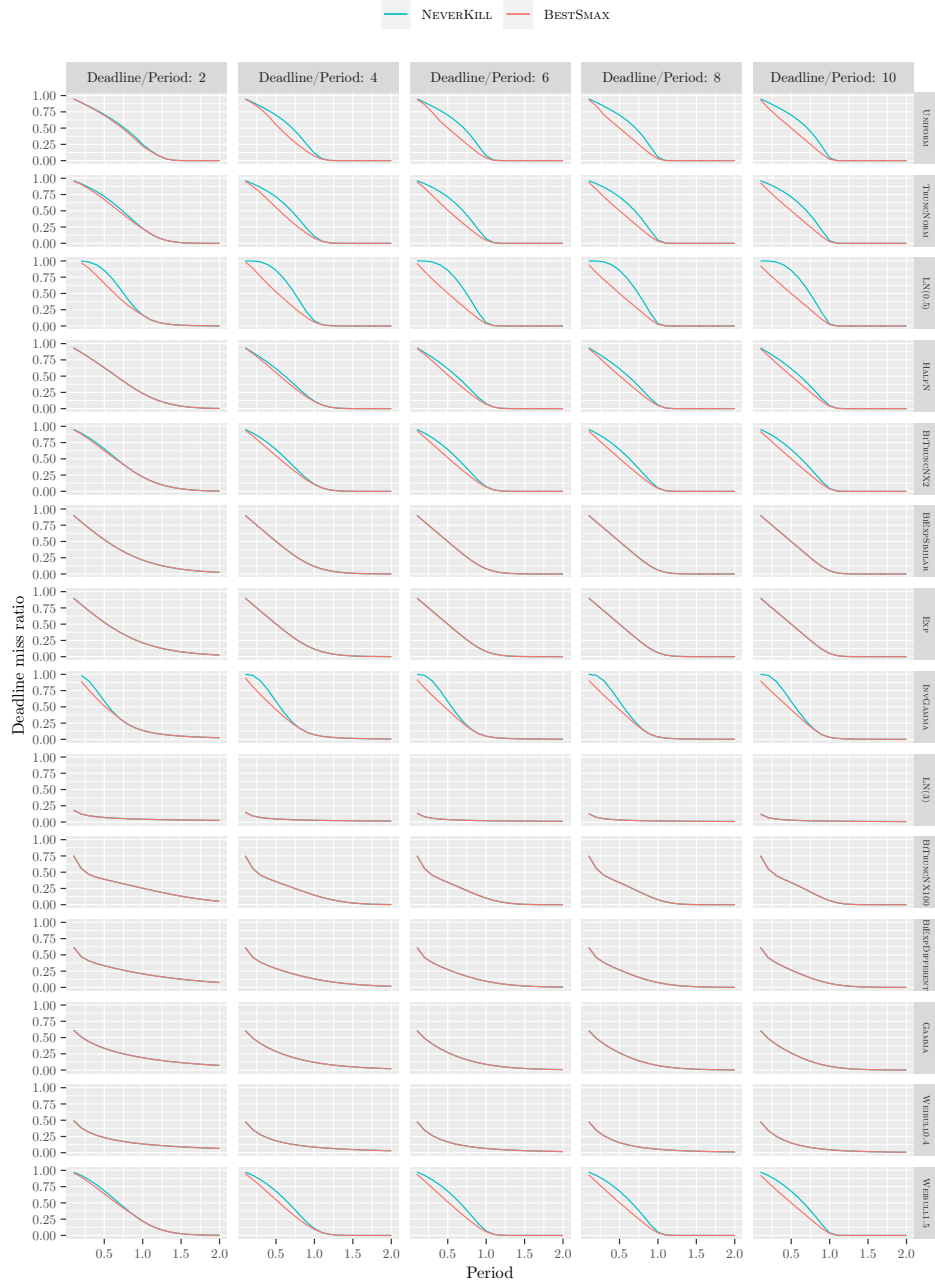


Figure 18: Absolute performance of job admission heuristics BINSMAX and NEVERKILL when the quantum length is 0.1 time unit. The choice of the values of the parameter s_{\max} is defined through the Markov chain approach, but the performance are assessed through simulations of 10^6 jobs.

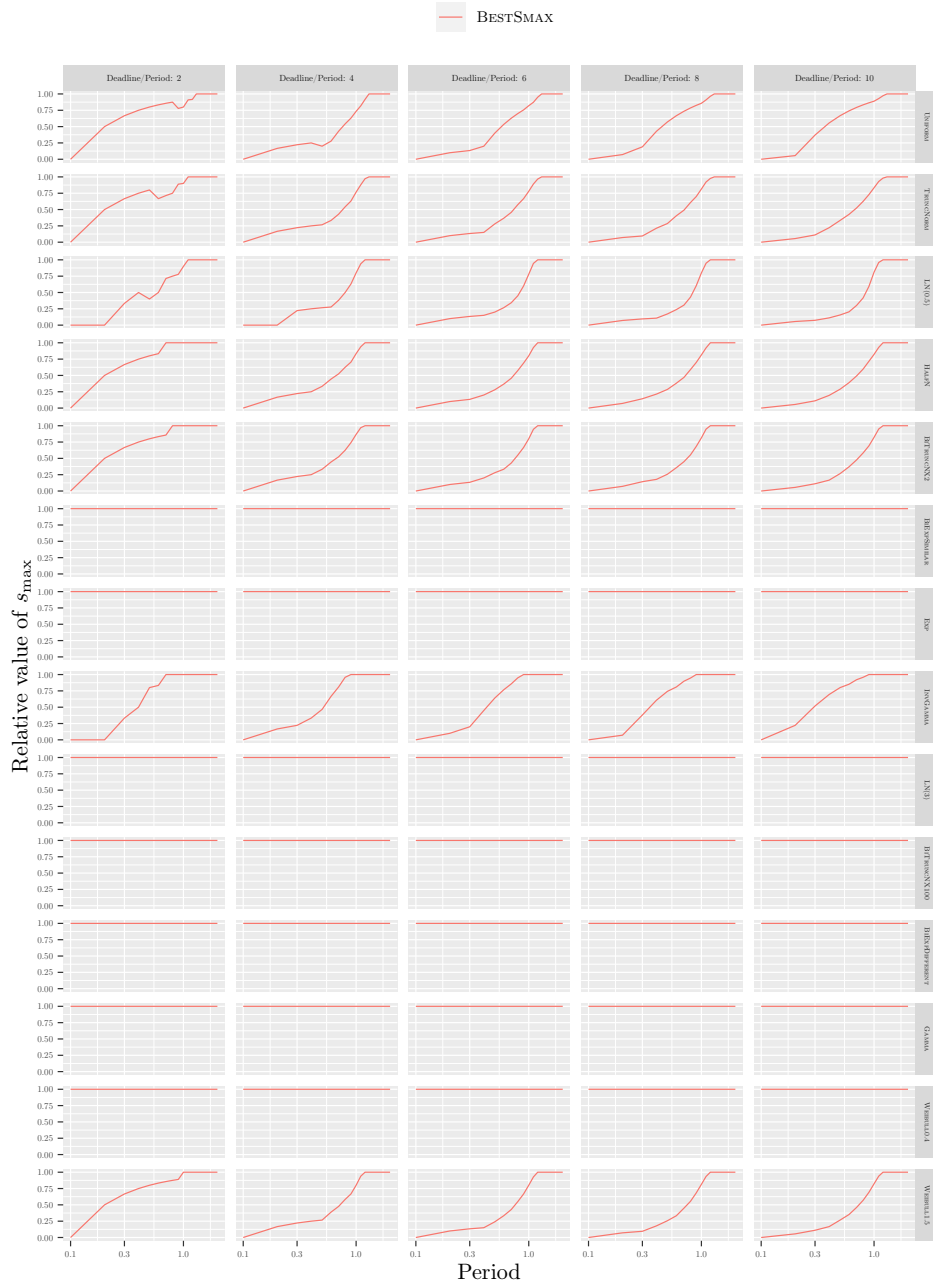


Figure 20: Best value for the s_{\max} threshold for job admission heuristic BESTSMAX expressed as a fraction of the maximum meaningful value $D - \tau$, when the quantum length is 0.1 time unit.

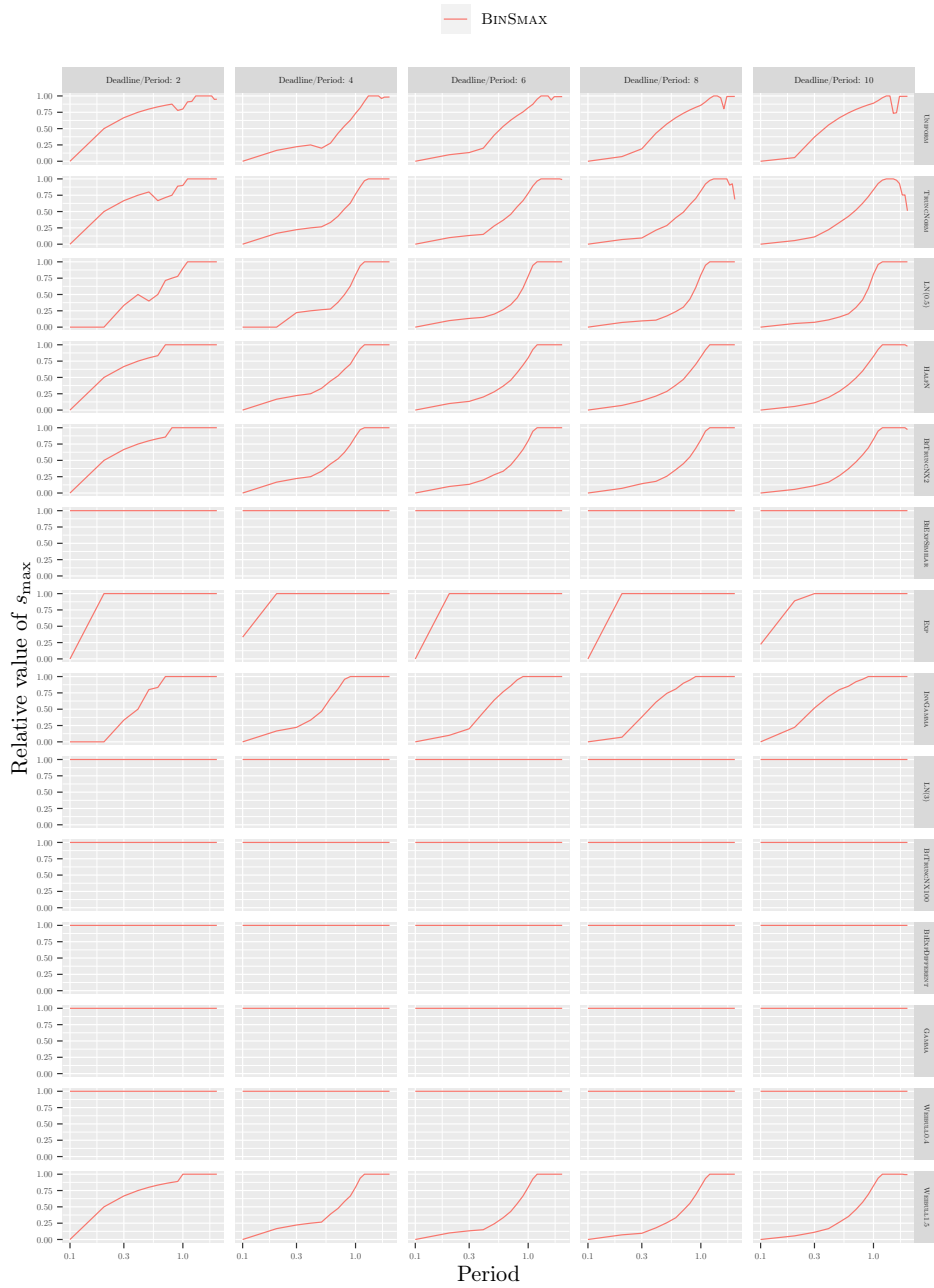
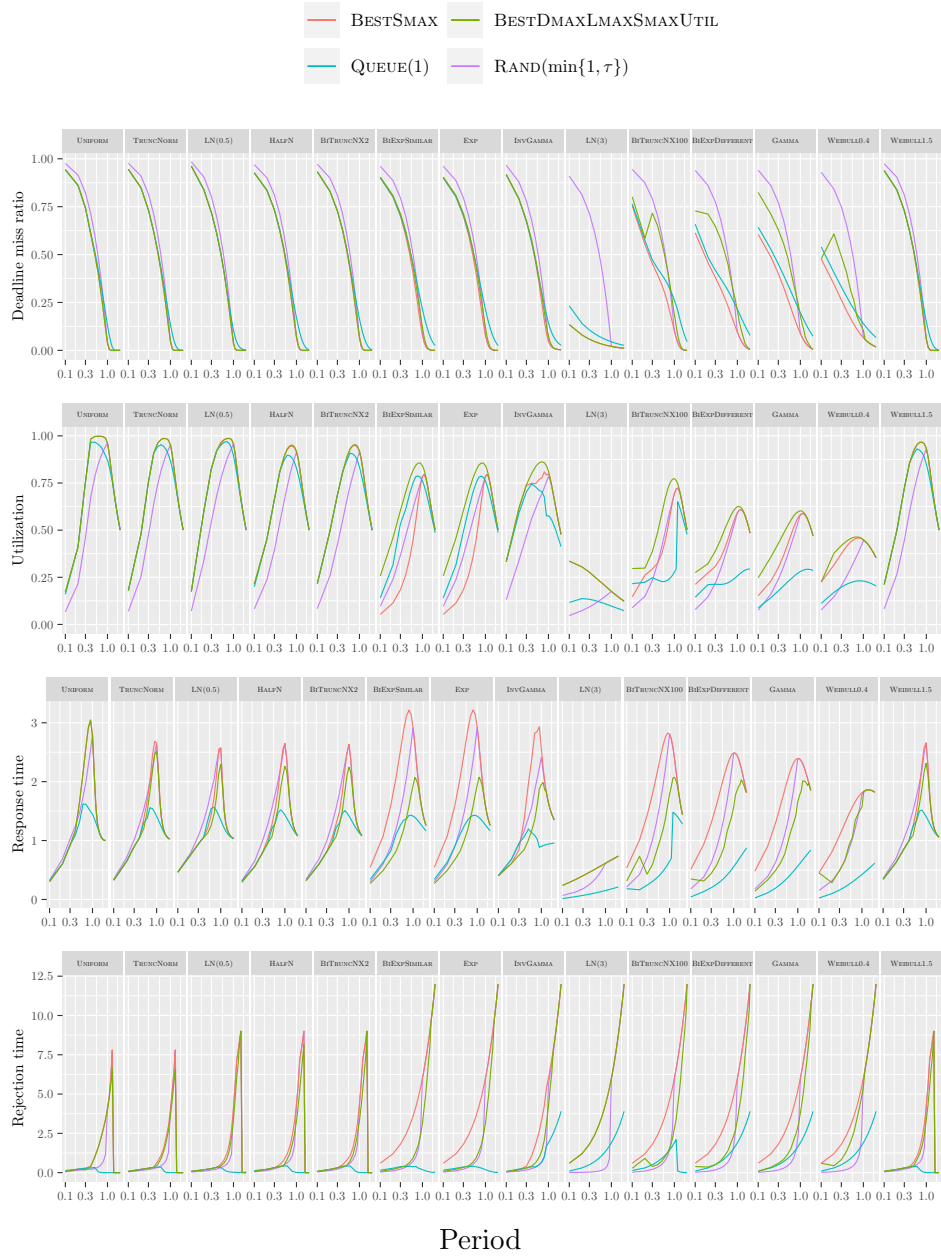


Figure 21: Best value for the s_{\max} threshold for job admission heuristic BINSMAX expressed as a fraction of the maximum meaningful value $D - \tau$, when the quantum length is 0.1 time unit.

Figure 22: Performance of different optimization criteria for different heuristics, while varying period τ and fixing $N = 4$.





**RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES**

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

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

ISSN 0249-6399