Stochastic Analysis of Periodic Real-Time Systems*

José Luis Díaz[†] Daniel F. García[†] Kanghee Kim[‡] Chang-Gun Lee[¶]
Lucia Lo Bello[§] José María López[†] Sang Lyul Min[‡] Orazio Mirabella[§]

Abstract

This paper describes a stochastic analysis method for general periodic real-time systems. The proposed method accurately computes the response time distribution of each task in the system, thus making it possible to determine the deadline miss probability of individual tasks, even for systems with maximum utilization factor greater than one. The method uniformly covers both fixed-priority scheduling (such as Rate Monotonic) as well as dynamic-priority scheduling (such as Earliest Deadline First) and can handle arbitrary relative deadlines and execution time distributions. The accuracy of the method is proven by comparing the results from the analysis with those obtained from simulations, as well as other methodologies in the literature.

1. Introduction

Traditional scheduling algorithms and analysis methods, such as processor utilization analysis [16, 11] and response time analysis [4, 19], focus on strict "hard" deadlines, by which a system is deemed schedulable only if every instance (called a job) of every task is guaranteed to meet its deadline. Although this deterministic timing guarantee is needed in hard real-time systems, it is too stringent for so-called soft real-time applications that require only a probabilistic guarantee that the deadline miss ratio of a task is below a given threshold. For such soft real-time applications, we need to relax the assumption that every instance of a task requires the worst-case execution time and analyze system behavior from a statistical point of view.

Progress has recently been made in the analysis of real-time systems under the stochastic assumption that jobs from a task require variable execution times. Research in this area can be categorized into two groups depending on the approach it takes to facilitate the analysis. The methods in the first group introduce a worst-case assumption to simplify the analysis (e.g., the critical instant assumption in Probabilistic Time Demand Analysis [18] and Stochastic Time Demand Analysis [7, 6]) or a restrictive assumption (e.g., the heavy traffic condition in the Real-Time Queueing Theory [12, 13]). Those in the second group, on the other hand, assume a special scheduling model that provides isolation between tasks so that each task can be analyzed independently of other tasks in the system (e.g., the reservation-based system addressed in [1] and Statistical Rate Monotonic Scheduling [2]).

In this paper, we propose a stochastic analysis method that does not introduce any worst-case or restrictive assumptions into the analysis, and is applicable to general priority-driven real-time systems. The method is general in the sense that it covers general priority-driven systems including both fixed-priority systems such as RM [16] and DM [15] and dynamic-priority systems such as EDF [16] (First In First Out is also covered since it is considered a special case of EDF where all the jobs have a constant relative deadline). The analysis method can handle any periodic task set consisting of tasks with arbitrary relative deadlines (including relative deadlines greater than the periods) and arbitrary execution time distributions.

The analysis method is based on Markov process modeling, which enables us to reason probabilistically about the steady-state behavior of the system even in the case of possible overload. It provides both analytical and numerical solutions for the deadline miss probabilities of tasks by

*The author names are listed in alphabetical order since this paper is a collaborative work based on two different papers submitted to the conference separately. One paper was written by José Luis Díaz, José María López and Daniel F. García, and the other was written by Kanghee Kim, Lucia Lo Bello, Chang-Gun Lee, Sang Lyul Min, and Orazio Mirabella.

† José Luis Díaz, José María López and Daniel F. García ({jdiaz,chechu,daniel}@atc.uniovi.es), *Departamento de Informática, Universidad de Oviedo* (33204, Gijón, Spain)

‡ Kanghee Kim (khkim@archi.snu.ac.kr) and Sang Lyul Min (symin@dandelion.snu.ac.kr), *School of Computer Science and Engineering, Seoul National University* (Shillim-dong San 56-1, Kwanak-Gu, Seoul, 151-742 Korea), supported in part by the Ministry of Science and Technology under the National Research Laboratory program and by the Ministry of Education under the BK21 program

§ Lucia Lo Bello and Orazio Mirabella ({llobello,omirabel}@diit.unict.it), *Dipartimento di Ingegneria Informatica e delle Telecomunicazioni, Facoltà di Ingegneria, Università di Catania* (Viale A. Doria 6, 95125 Catania, Italy)

¶ Chang-Gun Lee (cglee@ee.eng.ohio-state.edu), *Department of Electrical Engineering, Ohio State University* (2015 Neil Avenue, Columbus, OH 43210, U.S.A.)

computing the complete probability function (PF) of the response time of each task.

The rest of the paper is organized as follows. In Section 2, the related work is described in detail. In Section 3, the system model assumed and the notations used throughout the paper are given. Section 4 describes the stochastic analysis method we propose and also explains various approximation techniques to reduce its computational overheads. In Section 5, we compare experimental results from the proposed analysis method with those obtained from simulations and other methodologies. Finally, in Section 6, we conclude the paper with directions for future research.

2. Related work

Several studies have addressed the variability of task execution times even in deterministic schedulability analysis. For example, a multiframe model was proposed by Mok and Chen [17] in which the execution time of a task may vary greatly from one instance to another assuming that this variation follows a known pattern. The pattern is given as a finite list of numbers, and the execution times of successive instances are generated from the list. From this model, new utilization bounds which improve those of Liu and Layland [16] are derived for fixed-priority preemptive scheduling. However, since this model is aimed at providing a deterministic timing guarantee, it is still pessimistic.

In the attempt to provide probabilistic guarantees to real-time tasks with variable execution times, some reservation-based models that provide isolation between tasks have been studied. These models include the reservation-based system addressed by Abeni and Buttazzo [1] and a modification of Rate Monotonic Scheduling proposed by Atlas and Bestavros [2], which is called Statistical Rate Monotonic Scheduling (SRMS). Both assume reservation-based scheduling algorithms so that the analysis can be performed as if each task had a dedicated (virtual) processor. That is, for each task, a guaranteed budget of processor time is provided in every period [1] or super-period (the period of the next low-priority task which is assumed to be an integer multiple of the period of the task in SRMS) [2]. Therefore, the deadline miss probability of a task can be analyzed independently of other tasks assuming the guaranteed budget. However, the stochastic analysis methods developed for these systems are not applicable to general priority-driven systems due to the adoption of non-priority-driven scheduling algorithms or the modification of the original priority-driven scheduling rules.

A more general approach is to model the system as a single server queue, and try to apply the results of classical queueing theory. The real-time system we are trying to analyze is made up of periodic independent tasks scheduled by a preemptive priority-driven scheduler on a uniprocessor. Apparently, this system is a multiclass queue in which each class has deterministic inter-arrival times and arbitrary service times (D/G/1). However, this model cannot account for the relative phases of the tasks, which influence the statistical distribution of the response times. Furthermore, the few results available for multiclass queues assume specific distributions (Poisson, normal, etc.) for the inter-arrival times. The case of general distributions of inter-arrival times is usually addressed under "heavy-traffic" conditions (i.e., the average utilization is close to one) and, even in this case, the results are not valid for deterministic inter-arrival times, because this is a special case of a general distribution.

One extension of the classical queueing theory to deal with real-time issues was proposed in the Real-Time Queueing Theory [12]. This analysis method is flexible in that it is not limited to a particular scheduling algorithm and can be extended to real-time queueing networks. However, the analysis method assumes that the system is under heavy-traffic conditions. In addition, it only considers one class of clients, i.e., the interarrival times and execution times are identically distributed for all the tasks. This model does not fit well with the periodic task model.

Another approach to the statistical characterization of real-time systems is to extend an existing response time analysis, substituting the fixed execution times with random variables. Following this approach, Tia et al. [18] proposed the Probabilistic Time Demand Analysis (PTDA), which substitutes the sums of fixed execution times in the Time Demand Analysis [14] with convolutions of probability functions (PFs). In this way they can obtain the PF of the response time of a task assuming the worst-case scenario on the task release times, i.e., the critical instant. The analysis is restricted to the first activation of the task, since the deadlines cannot be greater than the periods. This assumption is also made in the Stochastic Time Demand Analysis (STDA) by Gardner [6], which extends the PTDA to cover systems where the deadlines may be greater than the periods. In this analysis, the probability of deadline misses is computed for each job released in the first in-phase busy interval, and the maximum of these probabilities is chosen as an upper bound on the probability of deadline misses for the corresponding task. Both analyses are based on the sum of random variables, and thus the use of convolutions to determine its PF. However, the analyses cannot address systems where the maximum system utilization is greater than one. In this case, the assumption that the busy interval starting at the critical instant will contain the worst-case response time is no longer valid, so the deadline miss probabilties computed by the analyses are not actual bounds.

Recently, Kim et al. [9] analyzed the stochastic behavior of a dynamic-priority system combined with an overrun handling mechanism called *randomized dropping*. Although this stochastic analysis method was developed to

compute the deadline miss probabilities of tasks for such a system, it can still be used for a "pure" dynamic-priority system. By modeling the system as a Markov process, it computes the stationary response time distributions of all the jobs in a hyperperiod (which is defined as a period whose length is equal to the least common multiple of the periods of all the tasks) and thus the response time distributions of all the tasks. However, it deals with dynamic-priority systems only and the derivation of the Markov matrix is complicated due to the abstraction of *job groups* and *aggregated response times*.

In this paper, we present a broader approach, which starts from the ideas of [18], [6] and [9]. We provide a simple derivation method for the Markov matrix and both analytical and numerical solutions for the response time distributions of all the tasks. We also describe a generalized framework to deal with both fixed-priority and dynamic-priority systems extending the concept presented in [9]. This model can be more easily understood starting from a simpler one, in which the system is not seen as a set of periodic tasks, but as a set of jobs released in a given sequence. Based on the simpler model, a systematic method to compute the response time PF of any job is developed and precisely described in [5, 8]. In Section 4.1, we will summarize this method in the more general context of a periodic task system, and will provide an overview of the whole stochastic analysis. Finally, note that our stochastic analysis is useful not only for soft real-time systems, but also for so-called *probabilistic hard real-time systems* [3], where a probabilistic guarantee close to 100% suffices.

3. System model and notation

The system is modeled as a set of N independent periodic tasks $S = \{\tau_1, \tau_2, ..., \tau_N\}$, each task τ_i being modeled by $(T_i, \Phi_i, \mathcal{C}_i, D_i)$, where T_i is the period of the task, Φ_i its initial phase, $\mathcal{C}_i$ its execution time, and D_i its relative deadline. The execution time is a discrete random variable* with a known probability function (PF), denoted by $f_{\mathcal{C}_i}(\cdot)$, where $f_{\mathcal{C}_i}(c) = \mathbb{P}\{\mathcal{C}_i{=}c\}$. Since the value of $\mathcal{C}_i$ is bounded, its PF can be stored as a finite vector of values $[f_{\mathcal{C}_i}(C_i^{\min}), \ldots, f_{\mathcal{C}_i}(C_i^{\max})]$.

Each task gives rise to an infinite sequence of jobs, and we will denote the j-th job of task τ_i by $\Gamma_{i,j}$. The release time of job $\Gamma_{i,j}$ will be denoted by $\lambda_{i,j}$. This time is deterministic and equal to $\Phi_i + (j-1) \times T_i$. Each job requires an execution time which is a random variable whose distribution is given by $f_{\mathcal{C}_i}(\cdot)$, and it is assumed to be independent of other jobs of the same task and those of other tasks.

The scheduling policy we assume is a general priority-driven one that assigns each job $\Gamma_{i,j}$ a static priority and schedules jobs according to this priority. The scheduler guarantees that the running job is the one with the highest priority among the ready jobs. We are not concerned with the policy used to assign priorities to jobs, as long as they are assigned in a deterministic way. This model includes well-known policies such as *Rate Monotonic* (RM), *Deadline Monotonic* (DM) and *Earliest Deadline First* (EDF). For fixed-priority policies, we will use P_i to denote the priority assigned to task τ_i.

The response time of the job $\Gamma_{i,j}$ will be represented by $\mathcal{R}_{i,j}$. This is a random variable, which can take different values with different probabilities. In the next section we outline a procedure which allows us to find the probability of occurrence of each possible response time for a given job, i.e. the probability function (PF) of the response time: $f_{\mathcal{R}_{i,j}}(r) = \mathbb{P}\{\mathcal{R}_{i,j}{=}r\}$

From the job response time PFs, the response time PF for any task can be obtained as the average of the response time PFs of the jobs belonging to that task. The task response time PF provides the analyst with significant information about the stochastic behavior of the system. In particular, it can be used to compute the probability of deadline misses for each task. The deadline miss probability DMP_{τ_i} of task τ_i can be computed from its response time PF as follows:

$$DMP_{\tau_i} = \mathbb{P}\{\mathcal{R}_{\tau_i} > D_i\} = 1 - \mathbb{P}\{\mathcal{R}_{\tau_i} \leq D_i\}$$

4. Stochastic analysis

4.1. Overview

To compute the response time PF of each task, we have to know the response time PFs of the jobs belonging to it. However, since the number of jobs generated by a task may be infinite, it is not possible to consider all of them for computation of its response time PF. To address this problem, we observe that the arrival pattern of jobs within a hyperperiod is repeated for all the other hyperperiods. Thus, if some stochastic regularity is found at the hyperperiod level, we can restrict our analysis to a single hyperperiod, and say that the derived job response time PFs are applicable for other hyperperiods. In this case, the response time PF $f_{\mathcal{R}_{\tau_i}}(\cdot)$ of task τ_i is represented by the average of the response time PFs of all the jobs from the task in the hyperperiod. That is,

$$f_{\mathcal{R}_{\tau_i}}(r) = \frac{1}{m_i} \sum_{j=1}^{m_i} f_{\mathcal{R}_{i,j}}(r) \qquad (1)$$

where $m_i = T/T_i$ is the number of jobs from τ_i released in a hyperperiod of length T.

To address stochastic regularity in hyperperiods, we first define the *P-level backlog* observed at time t as the sum of the remaining execution times of all the jobs that have

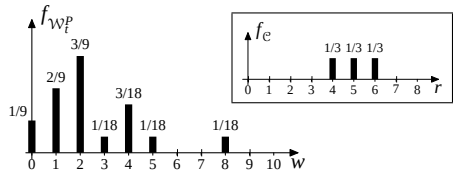
*Throughout this paper we use a calligraphic typeface to denote random variables, like $\mathcal{C}$, $\mathcal{R}$, etc.

priorities higher than or equal to P and are not completed up to the time t. This random quantity is denoted by $\mathcal{W}_t^P$. Then, we focus on the P-level backlog observed at the beginning of each hyperperiod k, denoted by $\mathcal{B}_k^P = \mathcal{W}_{kT}^P$, and investigate the stochastic process defined as the sequence of random variables $\{\mathcal{B}_1^P, \mathcal{B}_2^P, ..., \mathcal{B}_k^P, ...\}$. We prove that this stochastic process is a Markov chain. In addition, a stationary distribution of the P-level backlog $\mathcal{B}_k^P$ exists as long as the stability condition whereby the average system utilization is less than one is met. By deriving the Markov matrix that gives the transition probability $\mathbb{P}\{(\mathcal{B}_k^P = t_1) \rightarrow (\mathcal{B}_{k+1}^P = t_2)\}$ for any two states t_1 and t_2, we compute the exact stationary P-level backlog PF $f_{\mathcal{B}_k^P}(\cdot)$, observed at the beginning of the hyperperiod.*

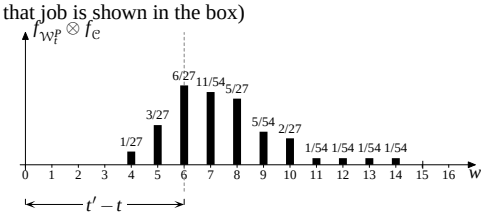
Once the stationary P-level backlog PF $f_{\mathcal{B}_k^P}(\cdot)$ is given, the stationary P-level backlog PF observed at any time within the hyperperiod can easily be calculated using the method explained in [5, 8]. Basically, by using two simple operations called *convolution* and *shrinking*, the P-level backlog PF $f_{\mathcal{W}_{t'}^P}(\cdot)$ at any time $t'(> t)$ can be calculated from the P-level backlog PF $f_{\mathcal{W}_t^P}(\cdot)$ at time t. For example, let us assume a simple scenario in which a job with a priority higher than or equal to P is released at time t and there is no further release of jobs with a priority higher than or equal to P in the interval $[t, t')$. In this case, the P-level backlog PF observed immediately after the release of the job, i.e., $f_{\mathcal{W}_{t+}^P}(\cdot)$, is obtained by performing convolution between $f_{\mathcal{W}_t^P}(\cdot)$ and the execution time PF of the job. Then, the P-level backlog PF $f_{\mathcal{W}_{t'}^P}(\cdot)$ at time t' is obtained by shrinking $f_{\mathcal{W}_{t+}^P}(\cdot)$, that is, shifting $f_{\mathcal{W}_{t+}^P}(\cdot)$ to the left $(t'-t)$ units and accumulating in the origin all the probability values defined for the non-positive time values. These operations are graphically shown through an example in Figure 1. Therefore, by iteratively applying convolution and shrinking to the stationary P-level backlog PF observed at the beginning of the hyperperiod, we can compute the stationary backlog PFs for all the jobs with priority P, observed at their release times.

In order to compute the response time of a job, it is necessary to know the *job-level backlog*, defined as the backlog due to jobs with priorities higher than or equal to the priority of that job, observed at a given time (usually the release time of the job). It is clear that, under a fixed-priority scheduling policy, the job-level backlog coincides with the P-level backlog, P being the priority of the job under consideration. However, for dynamic-priority policies such as EDF the method for obtaining the job-level backlog is different. In Section 4.4 we will deal with the general case.

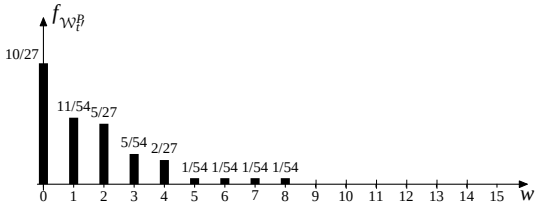
*It is possible to define a different hyperperiod length for each priority level P by computing the LCM only for tasks with priority higher than or equal to P. This would reduce the computational cost of the method, but we will not use this approach in the text, in the interest of clarity.



(a) At time t, just before the release of the job (the execution time PF for that job is shown in the box)



(b) At time t, just after the release of the job (convolution)



(c) At time $t' = t + 6$ (shrinking)

Figure 1. Example of the P-level backlog PF at two different times, separated by 6 units of time

For now it is sufficient to know that the job-level backlog can be obtained from the *system-level backlog*, defined as the backlog due to all the jobs released before a given time (this can be regarded as a 0-level backlog). In the general case, then, the job-level backlog is different from the P-level backlog, so we need to introduce a new notation to differentiate between them. We will represent by $\mathcal{V}_{i,j}$ the job-level backlog of the job $\Gamma_{i,j}$ present at its release time. Note that in the particular case of fixed-priority tasks, (such as RM) we can say that $\mathcal{V}_{i,j} = \mathcal{W}_{\lambda_{i,j}}^{P_i}$, P_i being the priority of the task τ_i to which the job belongs.

After the job-level backlog PFs of all the jobs in the hyperperiod have been computed, we compute their response time PFs. For each job $\Gamma_{i,j}$, the response time PF can easily be calculated, since the response time $\mathcal{R}_{i,j}$ is defined by the following equation

$$\mathcal{R}_{i,j} = \mathcal{V}_{i,j} + \mathcal{C}_i + \sum_{\Gamma_{k,l} \in H} \mathcal{C}_k \qquad (2)$$

where H is the set of all the jobs that may preempt $\Gamma_{i,j}$, i.e.,

the set of jobs released after time $\lambda_{i,j}$ with a priority higher than that of job $\Gamma_{i,j}$. Thus, the stationary response time PF $f_{\mathcal{R}_{i,j}}(\cdot)$ of $\Gamma_{i,j}$ can be computed from the stationary job-level backlog PF $f_{\mathcal{V}_{i,j}}(\cdot)$ and the execution time PF $f_{\mathcal{C}_i}(\cdot)$ of $\Gamma_{i,j}$ and the execution time PFs of the higher-priority jobs that preempt $\Gamma_{i,j}$.

The response time PF is computed as follows. Let Γ'_k be the k-th job in H (we assume H to be ordered by the release times), and λ'_k the time that has elapsed between the release of $\Gamma_{i,j}$ and that of Γ'_k. First, we calculate the response time PF not considering any possible preemptions by the higher-priority jobs by convolving the job-level backlog PF $f_{\mathcal{V}_{i,j}}(\cdot)$ and the execution time PF $f_{\mathcal{C}_i}(\cdot)$ of the job $\Gamma_{i,j}$. Then, iteratively, we calculate the response time PF that reflects all possible preemptions by Γ'_1, Γ'_2, ..., Γ'_k, ... by convolving the execution time PFs of Γ'_k's, in turn ($k = 1, 2, 3, ...$), into the response time PF of $\Gamma_{i,j}$. At each step, say k, the convolution is applied to the tail part of the response time PF being calculated, which is defined in the range (λ'_k, ∞), since the higher-priority job Γ'_k may affect $\Gamma_{i,j}$ only if $\Gamma_{i,j}$ executes up to the release time of Γ'_k. In other words, at step k, the response time PF of $\Gamma_{i,j}$ is calculated by (1) splitting the response time PF obtained at step $k-1$ into the tail part defined for the range (λ'_k, ∞) and the remaining head part, (2) convolving the tail part and the execution time PF $\mathcal{C}'_k$ of Γ'_k, and finally (3) merging the head part and the new tail part resulting from the convolution. Figure 2 shows this process graphically, for the k-th higher-priority job.

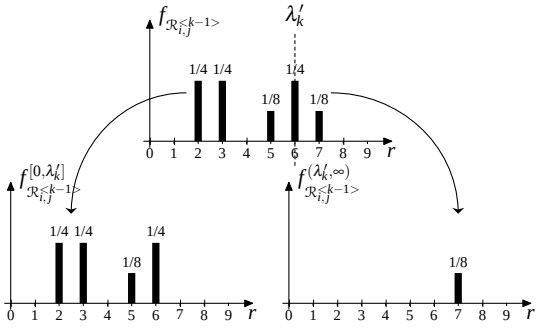
This process can also be more formally expressed through the following notation: let $f^{\mathbf{I}}(\cdot)$ be a partial PF defined as the part of the PF $f(\cdot)$ which takes values in the interval $\mathbf{I}$, as follows.

$$f^{\mathbf{I}}(r) = \begin{cases} f(r) & \text{if } r \in \mathbf{I} \\ 0 & \text{otherwise} \end{cases}$$

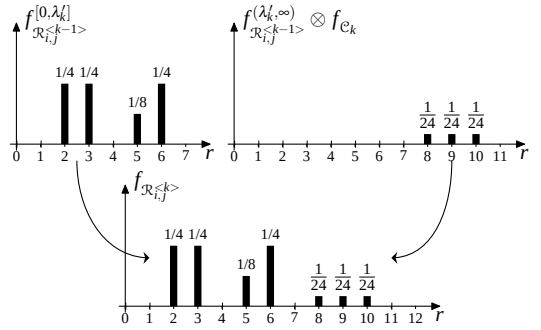
Also, let $\mathcal{R}_{i,j}^{<k>}$ be a random variable describing the response time of job $\Gamma_{i,j}$ that reflects all possible preemptions by the higher-priority jobs $\Gamma'_1, \Gamma'_2, \ldots, \Gamma'_k$ that can preempt $\Gamma_{i,j}$ ($\lambda'_1 \leq \lambda'_2 \leq \cdots \leq \lambda'_k$). Then the response time PF $f_{\mathcal{R}_{i,j}^{<k>}}(\cdot)$ can be calculated from the response time PF $f_{\mathcal{R}_{i,j}^{<k-1>}}(\cdot)$ and the execution time PF $f_{\mathcal{C}'_k}(\cdot)$ of Γ'_k as follows:

$$f_{\mathcal{R}_{i,j}^{<k>}}(r) = f_{\mathcal{R}_{i,j}^{<k-1>}}^{[0,\lambda'_k]}(r) + (f_{\mathcal{R}_{i,j}^{<k-1>}}^{(\lambda'_k,\infty)} \otimes f_{\mathcal{C}'_k})(r)$$

However, in order to obtain the deadline miss probability for each job $\Gamma_{i,j}$, it is not necessary to calculate the complete response time PF, because $\mathbb{P}\{\mathcal{R}_{i,j} > D_i\} = 1 - \mathbb{P}\{\mathcal{R}_{i,j} \leq D_i\}$, and the probability $\mathbb{P}\{\mathcal{R}_{i,j} \leq D_i\}$ can be computed from the part of the response time PF defined in the range [0, D_i]. If, at step k, we find that $D_i \leq \lambda'_k$ in the calculation of the response time PF, the calculation process can be stopped,



(a) "Splitting" the response time PF obtained in step $k-1$



(b) Construction of the response time PF for step k

Figure 2. Response time PF calculation

and the exact deadline miss probability can be computed. Moreover, the calculation process can also be stopped when the response time PF obtained at some step k reveals that the job $\Gamma_{i,j}$ has a null probability of being running up to the release time of the next higher-priority job Γ'_{k+1} (i.e., $f_{\mathcal{R}_{i,j}^{<k>}}^{(\lambda'_k,\infty)}(\cdot)$ is zero).

In the following subsection, we will describe how to compute the stationary P-level (or system-level) backlog PF observed at the beginning of the hyperperiod, and then explain how to deal with dynamic-priority systems such as EDF in the job-level backlog PF computation. In the following description, note that, whenever we omit the superscript P in $\mathcal{B}_k^P$, a priority level P is implicit in the notation.

4.2. Markovian modeling and stability

The backlog at the beginning of the k-th hyperperiod is a random variable, whose distribution, in general, is different for each k. So, the sequence of random variables $\{\mathcal{B}_k\}$ is a random process, which we will call the "backlog process". We will show that this process is a Markov chain.

The PF of $\mathcal{B}_k$ can be expressed in terms of the PF of

$\mathcal{B}_{k-1}$ using conditional probabilities:

$$\mathbb{P}\{\mathcal{B}_k{=}j\} = \sum_i \mathbb{P}\{\mathcal{B}_{k-1}{=}i\}\mathbb{P}\{\mathcal{B}_k{=}j \mid \mathcal{B}_{k-1}{=}i\} \quad (3)$$

Moreover, in our case, all hyperperiods receive the same sequence of jobs, in the same order, so the conditional probabilities $\mathbb{P}\{\mathcal{B}_k{=}j \mid \mathcal{B}_{k-1}{=}i\}$ do not depend on k. That is to say, $\mathbb{P}\{\mathcal{B}_k{=}j \mid \mathcal{B}_{k-1}{=}i\} = \mathbb{P}\{\mathcal{B}_1{=}j \mid \mathcal{B}_0{=}i\}$. Then, the PF of $\mathcal{B}_k$ depends only on the PF of $\mathcal{B}_{k-1}$, and not on those of $\mathcal{B}_{k-2}, \mathcal{B}_{k-3}, \ldots$. This is the "memoryless" property of a Markov chain.

In this way, we can write Equation (3) in matrix form:

$$\mathbf{b}_k = \mathbf{P}\,\mathbf{b}_{k-1} \quad (4)$$

where $\mathbf{b}_k$ is the PF of $\mathcal{B}_k$ in the form of an infinite column vector $\left[\mathbb{P}\{\mathcal{B}_k{=}0\}, \mathbb{P}\{\mathcal{B}_k{=}1\}, \ldots\right]^\intercal$, and $\mathbf{P}$ is the infinite Markov matrix, whose elements we will call $b_j(i)$, defined as:

$$\mathbf{P}(i,j) = b_j(i) = \mathbb{P}\{\mathcal{B}_1{=}i \mid \mathcal{B}_0{=}j\}$$

Under stability conditions, the Markov chain is ergodic, and in this case the PFs of the random variables $\{\mathcal{B}_k\}$ converge towards a single stationary distribution as k approaches infinity.

The stability condition depends on the *average system utilization* $\bar{U}$, which is defined as:

$$\bar{U} = \sum_{i=1}^{N} \frac{\bar{C}_i}{T_i} \quad (5)$$

$\bar{C}_i$ being the expected value of the execution time for the task τ_i.

Whenever $\bar{U} < 1$, the system is stable, and the sequence of random variables $\mathcal{B}_k$'s converges towards a single random variable, whose PF is a vector usually denoted by $\boldsymbol{\pi}$. This vector can be obtained as the unique solution of the equation $\boldsymbol{\pi} = \mathbf{P}\boldsymbol{\pi}$, with the additional restriction of $\sum \pi_i = 1$.

The condition $\bar{U} < 1$ assures the stability and convergence of the system, and thus the existence and uniqueness of $\boldsymbol{\pi}$. This fact can be proved using Markov theory and "drifting conditions", but it is simpler to approximate our model by a queue with a single server. A well-known result of queueing theory (see for example [10]) is that the system is stable if $\rho < 1$, ρ being a parameter of the queueing model, which can be shown to be equivalent to our $\bar{U}$.

In order to obtain the steady state PF of the backlog, the equation $\boldsymbol{\pi} = \mathbf{P}\boldsymbol{\pi}$ needs to be solved. This, in general, is not an easy task, because the matrix $\mathbf{P}$ is infinite. However, due to the nature of our model, the matrix has a regular structure, which will allow us to provide a general method to obtain an analytical expression for $\boldsymbol{\pi}$. Let us show the regularity present in $\mathbf{P}$.

Each column $\mathbf{P}(\cdot, j)$ in the matrix $\mathbf{P}$ is the PF of the backlog at the end of the first hyperperiod, if the initial backlog was j. Then, each column j in $\mathbf{P}$ can be calculated by setting the initial backlog equal to j, and using the algorithm presented in Section 4.1 to obtain the PF of the backlog at the end of the first hyperperiod. For each different initial backlog value, we will expect to obtain a different PF for the final backlog, and so we will expect each column in $\mathbf{P}$ to be different. However, there exists an initial backlog value, which we will call r, from which the PF of the final backlog is always the same, only "shifted" one position to the right. This means that after column r, all the columns in $\mathbf{P}$ repeat the same values, only shifted down by one position. The general form of matrix $\mathbf{P}$ is thus:

$$\mathbf{P} = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \overset{\overset{r}{\downarrow}}{\begin{pmatrix} b_0(0) & b_1(0) & b_2(0) & \ldots & b_r(0) & 0 & 0 & 0 \\ b_0(1) & b_1(1) & b_2(1) & \ldots & b_r(1) & b_r(0) & 0 & 0 \\ b_0(2) & b_1(2) & b_2(2) & \ldots & b_r(2) & b_r(1) & b_r(0) & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots & b_r(2) & b_r(1) & \ddots \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots & b_r(2) & \ddots \\ b_0(m_r) & b_1(m_r) & b_2(m_r) & \ldots & b_r(m_r) & \vdots & \vdots & \ddots \\ 0 & 0 & 0 & \ldots & 0 & b_r(m_r) & \vdots & \ddots \\ 0 & 0 & 0 & \ldots & 0 & 0 & b_r(m_r) & \ddots \\ 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \ddots \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}}$$

In particular, r is the maximum possible value of the idle time in any hyperperiod. It can be calculated as:

$$r = T + W^{\min} - \sum_i C_i^{\min} \quad (6)$$

where $W^{\min}$ is the backlog at the end of the first hyperperiod in which all the jobs are released, assuming that the initial backlog is zero and all the jobs require their minimum execution time. $W^{\min}$ is usually zero, unless most of the work is concentrated at the end of the hyperperiod.

Then, the r-th column of the Markov matrix $\mathbf{P}$ represents the PF of the final backlog in the case of the initial backlog being equal to the maximum possible idle time. In this case, the whole hyperperiod is busy, and so column r of $\mathbf{P}$ will be the convolution of the execution time PFs of all the jobs released in the hyperperiod, shifted $(T-r)$ units to the left. An analogous reasoning applies when the initial backlog is $(r+1)$, $(r+2)$, and so on, because in all these cases the whole hyperperiod is a busy period. This is the reason for the regularity of matrix $\mathbf{P}$ from column r onwards.

Moreover, m_r is the index of the last non-zero element of column r in $\mathbf{P}$, and thus represents the maximum possible backlog which could appear when the initial backlog is r. If the initial backlog is less than r, the final backlog will be less than m_r, and this is why the first r columns have zeros from row m_r onwards.

4.3. Solution approaches

Once the Markov matrix **P** has been derived, we can compute the stationary *P*-level (or system-level) backlog PF observed at the beginning of the hyperperiod with either an analytical or a numerical method. The analytical method gives the exact solution for the stationary *P*-level backlog PF while the numerical method gives approximated solutions. In the analytical method, we differentiate between the case where $U^{\max} < 1$ from general cases, since in this special case the *P*-level backlog PF is equal for all hyperperiods, so there is no need to perform the Markov process modeling described in the previous subsection. In this case, we compute the exact solution without deriving the Markov matrix. For the general case, on the other hand, we compute the exact solution by deriving a finite set of equations that can completely describe the infinite stationary backlog distribution. To reduce the computational overheads required to compute the exact solution, we also introduce some approximation methods, which make a trade-off between analysis accuracy and computational overheads.

4.3.1 Exact solution when $U^{\max} \leq 1$

We will define the maximum system utilization $U^{\max}$, as the total utilization of the system calculated using the worst-case execution times of the jobs:

$$U^{\max} = \sum_{i=1}^{N} \frac{C_i^{\max}}{T_i} \tag{7}$$

In the particular case where $U^{\max} < 1$, the maximum amount of work generated in a hyperperiod will not exceed the hyperperiod length. In this case, the backlog observed at the end of the hyperperiod cannot increase without bounds. In fact, it can be proved that the backlog present at the end of the first hyperperiod in which all the tasks were released at least once, will be repeated at the end of any subsequent hyperperiod. In particular, if at time $t = 0$ all the tasks are released in-phase, then no backlog will be present at the end of the first hyperperiod, and thus all subsequent hyperperiods will start with a zero initial backlog.

In these particular cases, the stationary PF of the backlog observed at the end of the hyperperiods can be obtained by simple calculation of the backlog PF at the end of the first hyperperiod in which all the tasks are released at least once. Moreover, the backlog PF obtained in this case has a finite number of points.

Note that when $U^{\max} < 1$, the system can be analyzed using classical response time analysis. Using the worst-case execution times of all the tasks, the worst-case response time can be calculated, and comparing this response time with each deadline, the feasibility of the system can be determined. However, if a profile of the execution times is used instead of a single worst-case value, the probability of deadline misses can be found, and, if this probability is small enough for its application, the system could still be feasible. An example of the benefits of a statistical approach to the problem is presented in [5]

Task	T_i	Priority	$\mathcal{C}_i$
τ_1	4	High	$\{1,2\}$ with equal probability
τ_2	6	Low	$\{2,3,4\}$ with prob 0.2, 0.3 and 0.5

Table 1. A system example with $U^{\max} > 1$

4.3.2 Exact solution for the general case

Taking advantage of the regular structure of **P**, we present a method for finding the complete stationary PF of the backlog, denoted by $\boldsymbol{\pi}$. Since this distribution has an infinite number of points, what we will obtain is the exact value of some starting points, and then the expression in closed form for the rest of the points.

The general form of $\boldsymbol{\pi}$ consists of a set of initial points, whose values depend on the parameters of the system, followed by an infinite tail which approaches zero in a exponential way. Actually, the tail is a sum of exponential functions, whose parameters depend only on convolution of all the execution times for all the jobs in the system. We call this solution "analytical" in the sense that a closed form of the solution is found. However, to obtain this solution, the method requires the roots of a polynomial to be found, and some numerical methods will be required nevertheless.

In order to make the method more understandable, we will introduce an example system and solve it "by hand", giving indications about how the example can be generalized, instead of presenting a formal development of the general solution.

Let us consider the system shown in Table 1. For this system, the hyperperiod is 12. Task τ_1 is released three times, and τ_2 twice within the hyperperiod.

We will obtain the stationary distribution for the low-priority-level backlog, i.e., the PF of the backlog present at the beginning of any hyperperiod in the distant future. To do so, we need to construct the Markov matrix **P**. This is done by computing the PF of the backlog at time $T = 12$, for different initial backlogs $0, 1, 2, \ldots$ up to r. Each of these PFs will be a column in **P**. In this example, the maximum possible idle time in a period, r, is 5. The resulting Markov

matrix is:

$$\mathbf{P} = \begin{array}{c} \\ m_r \end{array} \overset{r}{\begin{pmatrix} 0.8375 & 0.595 & 0.3275 & 0.13125 & 0.035 & 0.005 & - & \cdots \\ 0.13125 & 0.2425 & 0.2675 & 0.19625 & 0.09625 & 0.03 & 0.005 & \ddots \\ 0.03125 & 0.13125 & 0.2425 & 0.2675 & 0.19625 & 0.09625 & 0.03 & \ddots \\ - & 0.03125 & 0.13125 & 0.2425 & 0.2675 & 0.19625 & 0.09625 & \ddots \\ - & - & 0.03125 & 0.13125 & 0.2425 & 0.2675 & 0.19625 & \ddots \\ - & - & - & 0.03125 & 0.13125 & 0.2425 & 0.2675 & \ddots \\ - & - & - & - & 0.03125 & 0.13125 & 0.2425 & \ddots \\ - & - & - & - & - & 0.03125 & 0.13125 & \ddots \\ - & - & - & - & - & - & 0.03125 & \ddots \\ - & - & - & - & - & - & - & \ddots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots \end{pmatrix}}$$

As expected, the numbers in column r are repeated in the following columns, shifted one position down. The index of the last non-zero element in the r-th column is what we called m_r in Section 4.2, and its value is 7 in this example. Matrix $\mathbf{P}$ always presents another regularity: from row m_r+1 onwards, the values in each row are the same, only shifted one position to the right. Moreover, these values are the same as the coefficients of the r-th column. In effect, for this example this kind of regularity starts from row 2, but in general it is only guaranteed for row (m_r+1).

Equation $\boldsymbol{\pi} = \mathbf{P}\boldsymbol{\pi}$ can be "developed" and gives rise to an infinite set of equations with infinite unknowns, which are the component π_i's of $\boldsymbol{\pi}$. This infinite set of equations can be divided into two subsets: a subset comprising the first (m_r+1) equations, and a second subset made up of the remaining equations. The last subset defines a recurrence relation between the components of $\boldsymbol{\pi}$.

In our example, the first (m_r+1) equations are:

$$\begin{aligned} \pi_0 &= 0.8375\pi_0 + 0.595\pi_1 + 0.3275\pi_2 + 0.13125\pi_3 \\ &\quad + 0.035\pi_4 + 0.005\pi_5 \\ &\vdots \\ \pi_7 &= 0.03125\pi_5 + 0.13125\pi_6 + 0.2425\pi_7 + 0.2675\pi_8 \\ &\quad + 0.19625\pi_9 + 0.09625\pi_{10} + 0.03\pi_{11} + 0.005\pi_{12} \end{aligned}$$

This makes a system of 8 equations with 13 unknowns. In general, there will be (m_r+1) equations with (m_r+r+1) unknowns. The remaining equations, provided by rows m_r+1 and following, have a general form, due to their regularity. The general form, for $j \geq m_r+1 = 8$, is

$$\begin{aligned} \pi_j &= 0.03125\pi_{j-2} + 0.13125\pi_{j-1} + 0.24250\pi_j + \\ &\quad + 0.26750\pi_{j+1} + 0.19625\pi_{j+2} + 0.09625\pi_{j+3} \\ &\quad + 0.03000\pi_{j+4} + 0.00500\pi_{j+5} \end{aligned}$$

Finding π_{j+5} we obtain the following recurrence relationship, which holds for $j \geq 8$:

$$\begin{aligned} \pi_{j+5} = &-6.25\pi_{j-2} - 26.25\pi_{j-1} + 151.5\pi_j - 53.5\pi_{j+1} \\ &- 39.25\pi_{j+2} - 19.25\pi_{j+3} - 6\pi_{j+4} \end{aligned} \tag{8}$$

The recurrence relation of Equation (8) can be put in matrix form:

$$\mathbf{Q}_{j+1} = \mathbf{A}\,\mathbf{Q}_j \qquad j \geq 8$$

where

$$\mathbf{Q}_j = (\pi_{j-2}, \pi_{j-1}, \pi_j, \pi_{j+1}, \pi_{j+2}, \pi_{j+3}, \pi_{j+4})^\intercal$$

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -6.25 & -26.25 & 151.5 & -53.5 & -39.25 & -19.25 & -6 \end{pmatrix}$$

The elements of the last row of matrix $\mathbf{A}$ are the coefficients of Equation (8). Note that they are easily obtained from = the r-th column of $\mathbf{P}$. Using the matrix form, it is easy to see that $\mathbf{Q}_9 = \mathbf{A}\,\mathbf{Q}_8$, $\mathbf{Q}_{10} = \mathbf{A}^2\,\mathbf{Q}_8$, and in general, for $n \geq 8$, $\mathbf{Q}_n = \mathbf{A}^{n-8}\,\mathbf{Q}_8$.

Since $\mathbf{A}$ is diagonalizable, we can write $\mathbf{A} = \mathbf{V}^{-1}\mathbf{D}\,\mathbf{V}$, where $\mathbf{V}^{-1}$ is the matrix whose columns are $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_7$, the eigenvectors of $\mathbf{A}$; $\mathbf{D}$ is the diagonal matrix whose elements are $\lambda_1, \lambda_2, \ldots, \lambda_7$, the eigenvalues of $\mathbf{A}$, and $\mathbf{V}$ is the inverse matrix of $\mathbf{V}^{-1}$. Once diagonalized, the operation $\mathbf{A}^{n-8}$ is easy to compute, leading to the equation:

$$\begin{aligned} \mathbf{Q}_n &= \mathbf{V}^{-1}\mathbf{D}^{n-8}\mathbf{V}\,\mathbf{Q}_8 \\ &= C_1\lambda_1^{n-8}\mathbf{v}_1 + C_2\lambda_2^{n-8}\mathbf{v}_2 + C_3\lambda_3^{n-8}\mathbf{v}_3 + C_4\lambda_4^{n-8}\mathbf{v}_4 \\ &\quad + C_5\lambda_5^{n-8}\mathbf{v}_5 + C_6\lambda_6^{n-8}\mathbf{v}_6 + C_7\lambda_7^{n-8}\mathbf{v}_7 \end{aligned} \tag{9}$$

where

$$(C_1C_2C_3C_4C_5C_6C_7)^\intercal = \mathbf{V}\cdot(\pi_6\pi_7\pi_8\pi_9\pi_{10}\pi_{11}\pi_{12})^\intercal$$

The characteristic polynomial of $\mathbf{A}$ is very easy to find, even without obtaining $\mathbf{A}$, directly from the elements of column r in $\mathbf{P}$. It can be proved that its general form is:

$$f(\lambda) = \Big(\sum_{i=0}^{m_r} b_r(i)\lambda^{m_r-i}\Big) - \lambda^{m_r-r} = 0 \tag{10}$$

Note that the degree of this polynomial is m_r, which can be proved to be equal to $T(U^{\max} - U^{\min}) + W^{\min}$.

In our example, the roots of this polynomial are $\lambda_1 = (-3.2976 + 1.825i)$, $\lambda_2 = (-3.2976 - 1.825i)$, $\lambda_3 = (-0.3099 + 3.0755i)$, $\lambda_4 = (-0.3099 - 3.0755i)$, $\lambda_5 = 1$, $\lambda_6 = 0.3476$ and $\lambda_7 = -0.1325$. Some of these have modulus greater than or equal to one. Looking at Equation (9) it can be seen that, if all $C_i \neq 0$, then $\pi_n \to \infty$ as $n \to \infty$, due to these eigenvalues, and the sum of the components of $\boldsymbol{\pi}$ will be infinite. This is not the case, because the Markov

chain is positive and thus the stationary solution has to be summable. This implies that the coefficients C_1, C_2, C_3, C_4 and C_5 which multiply these eigenvalues in Equation (9), must be equal to zero. This condition gives rise to five new equations.

In general, in the case of stability, the polynomial in Equation (10) has r roots with modulus greater than or equal to 1, and so it always provides r additional equations. This fact can be proved by applying Rouche's Theorem

To summarize, we now have a linear system with 13 equations, 8 from the first m_r rows of $\mathbf{P}$ and 5 from the condition of some C_i's being zero, and 13 unknowns $(\pi_0, \ldots, \pi_{12})$. Nevertheless, the reader can check that the equation derived from $C_5 = 0$ is a linear combination of the others, and can be removed. In the general case, the equation derived from the eigenvalue 1, is always a linear combination of the others. Following the described method, we will end up with a system of $(m_r + r)$ equations and $(m_r + r + 1)$ unknowns. Since the number of unknowns is one more than the number of equations, we can put each of the first 12 components as a linear function of the first component π_0. We will not write these expressions here for the sake of brevity. The coefficient C_i's are also a linear function of π_0. In this example we obtain $C_6 = -0.00136\,\pi_0$, $C_7 = -1.5057 \cdot 10^{-6}\,\pi_0$ (the remaining C_i's are zero, guaranteeing the summability of $\boldsymbol{\pi}$). If we use these values in Equation (9) we find the following general expression for any component of $\boldsymbol{\pi}$, valid for $n \geq 8$

$$\begin{aligned}\mathbf{Q}_n = &-0.00136\pi_0\mathbf{v}_6(0.3474)^{n-8} \\ &-1.5057\cdot 10^{-6}\pi_0\mathbf{v}_7(-0.1325)^{n-8}\end{aligned} \tag{11}$$

Note that $\mathbf{v}_6$ and $\mathbf{v}_7$ are two eigenvectors, which were calculated directly from matrix $\mathbf{A}$. So, the only unknown in the above equation is π_0. For any value of π_0, we obtain a complete vector $\boldsymbol{\pi}$, which has the property $\boldsymbol{\pi} = \mathbf{P}\boldsymbol{\pi}$. However, only one of these possible vectors is a PF (has a sum equal to 1). So, as a final condition, we impose $\sum_{i=0}^{\infty}\pi_i = 1$, and from this we can determine the required value for π_0. This equation is easy to solve, despite the infinite summation, because the expression for π_i is a convergent sum of exponentials, for $i > 6$.

Solving this sum and equating it to 1, we find the value of π_0 for this example, which is $\pi_0 = 0.738872$. From this, all the remaining values of $\boldsymbol{\pi}$ can be computed. The first 13 components are obtained directly from the system of equations, and their values are shown in the last column of Table 2, rounded to the sixth decimal. The remaining components are calculated from the formula given in Equation (11), which, after substituting the values of π_0, $\mathbf{v}_6$ and $\mathbf{v}_7$, and taking one of the components of vector $\mathbf{Q}_n$, leads to:

$$\pi_n = 10^{-4}\left(9.4311\times 0.3474^{n-6} + 0.011\times(-0.1325)^{n-6}\right)$$

valid for $n > 6$.

	$\mathcal{B}_0$	$\mathcal{B}_1$	$\mathcal{B}_2$	$\mathcal{B}_3$	$\mathcal{B}_5$	$\mathcal{B}_{10}$	$\mathcal{B}_{20}$	$\mathcal{B}_\infty$
0	1	0.837500	0.789734	0.768523	0.750897	0.740816	0.738968	0.738872
1	–	0.131250	0.150109	0.155394	0.158160	0.158899	0.158919	0.158917
2	–	0.031250	0.050976	0.059129	0.065050	0.067794	0.068186	0.068203
3	–	–	0.008203	0.013632	0.018639	0.021485	0.021964	0.021987
4	–	–	0.000977	0.002906	0.005524	0.007464	0.007850	0.007869
5	–	–	–	0.000385	0.001372	0.002430	0.002690	0.002705
6	–	–	–	0.000030	0.000299	0.000779	0.000934	0.000944
7	–	–	–	–	0.000053	0.000238	0.000321	0.000328
8	–	–	–	–	0.000007	0.000069	0.000110	0.000114
9	–	–	–	–	0.000000	0.000019	0.000037	0.000040
10	–	–	–	–	0.000000	0.000005	0.000013	0.000014
11	–	–	–	–	–	0.000000	0.000004	0.000005
12	–	–	–	–	–	0.000000	0.000001	0.000001
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 2. Backlog PF convergence

4.3.3 Approximations

Truncation of matrix P One possible approximation technique to compute the stationary system-level backlog PF is to truncate the Markov matrix $\mathbf{P}$ to a sufficiently large square matrix $\mathbf{P}'$, which was first introduced in [1]. Assuming that the infinite stationary distribution $\boldsymbol{\pi}$ can be modeled with a finite vector $\boldsymbol{\pi}' = [\pi'_0, \pi'_1, \pi'_2, ..., \pi'_n]$, we can lead to the following equation from $\boldsymbol{\pi} = \mathbf{P}\boldsymbol{\pi}$:

$$\boldsymbol{\pi}' = \mathbf{P}'\boldsymbol{\pi}'$$

where $\mathbf{P}'$ is an (n+1)-by-(n+1) matrix consisting of the component $\mathbf{P}(i,j)$'s ($0 \leq i, j \leq n$) of the Markov matrix $\mathbf{P}$. The resulting equation is an eigenvector problem, from which we can calculate the approximated solution $\boldsymbol{\pi}'$ with a numerical method. Among the calculated eigenvectors, we can choose as the approximated solution an eigenvector whose corresponding eigenvalue is equal to or sufficiently close to 1. In order to obtain a good approximation of the stationary distribution $\boldsymbol{\pi}$, the truncation point n should be increased as much as possible, which makes the eigenvalue converge to 1. Note that, by choosing an appropriate truncation point, we can achieve a trade-off between analysis accuracy and the computational overheads required to solve the corresponding eigenvector problem. The choice of a convenient truncation point is an open issue.

Iterative approximation Another approximation technique to obtain the stationary system-level backlog PF, which does not require derivation of the Markov matrix $\mathbf{P}$, is the simple iteration of the algorithm which computes the PF of the backlog at the end of the hyperperiod. For the example presented in Section 4.3.2, the results of successive iterations are shown in Table 2. It can be seen that each point of the PF converges towards the analytical solution (given in the last column). It can be proved that the convergence is geometrically ergodic. However, it is not known in advance how many iterations will be necessary to make the backlog PF "close enough" to the stationary distribution. It is clear that the rate of convergence depends on how close $\bar{U}$ is to 1, becoming slower as $\bar{U}$ approaches 1. However, we

have not yet found a bound, in terms of $\bar{U}$, of the number of iterations required by the iterative method.

Another important point to consider is the effect of introducing zero as the initial backlog for the system. In the steady state the initial backlog is a random variable which can take non-zero values. Thus, the response times in the steady state will be *worse* than those calculated for the first hyperperiod. Indeed, they will be worse than that calculated for any hyperperiod. Using zero as the initial backlog will lead to optimistic probabilities of deadline misses, so design decisions based on the iterative method should be taken carefully.

4.4. Extension to dynamic-priority systems

Although dynamic-priority systems seem considerably different from fixed-priority systems, there exists one similarity. The similarity is that, in a hyperperiod, there always exists at least one job that always takes the system-level backlog $\mathcal{W}_{\lambda_{i,j}}$ observed at its release time $\lambda_{i,j}$ as its job-level backlog $\mathcal{V}_{i,j}$, i.e., $\mathcal{V}_{i,j} = \mathcal{W}_{\lambda_{i,j}}$. Such a job is called a *ground job*, and has a lower priority than all the jobs released before its release time $\lambda_{i,j}$. Thus, once the stationary system-level backlog PF $f_{\mathcal{B}_k^0}(\cdot)$ observed at the beginning of the hyperperiod is given, the stationary job-level backlog PF $f_{\mathcal{V}_{i,j}}(\cdot)$ of every ground job $\Gamma_{i,j}$ can be calculated as explained in Section 4.1, by iteratively applying convolution and shrinking to $f_{\mathcal{B}_k^0}(\cdot)$ for each job released between the beginning of the hyperperiod and the release time $\lambda_{i,j}$ of the job $\Gamma_{i,j}$ (for a fixed-priority system, this statement means that the stationary job-level backlog PF of every job with priority P, which is considered a ground job at the priority level P, can be calculated from $f_{\mathcal{B}_k^P}(\cdot)$).

Therefore, the difference between dynamic-priority and fixed-priority systems lies in how to compute the job-level backlog PFs of *non-ground jobs*. For fixed-priority systems, this problem is solved by considering the higher priority level that each non-ground job belongs to. Since any job classified as a non-ground job at a lower priority level is bound to become a ground job at the priority level that is equal to the priority of the job, its job-level backlog PF can be calculated from the stationary system-level backlog PF obtained at that priority level. However, for dynamic-priority systems, we can avoid such an iterative analysis, and compute the job-level backlog PF of the non-ground job from that of a preceding ground job. The preceding ground job is the last ground job that is released before the non-ground job and has a higher priority, which is called the *base ground job* for the non-ground job. Since the job-level backlog PF of any ground job can be calculated from the system-level backlog PF $f_{\mathcal{B}_k^0}(\cdot)$ observed at the beginning of the hyperperiod, this means that the job-level backlog PFs of all the jobs including the non-ground jobs in the hyperperiod can be calculated from the system-level backlog PF $f_{\mathcal{B}_k^0}(\cdot)$. For example, consider the task set shown in Figure 3. This task set consists of two tasks τ_1 and τ_2. The periods of τ_1 and τ_2 are 30 and 50, respectively, and the relative deadline D_i of each task is equal to T_i. The phase Φ_i's of both tasks are 0.

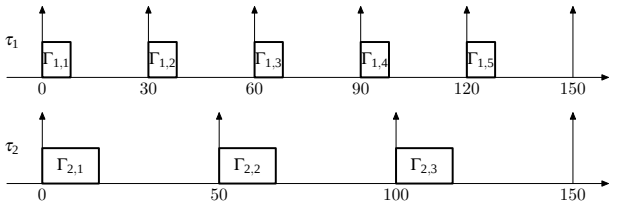


Figure 3. A task set example

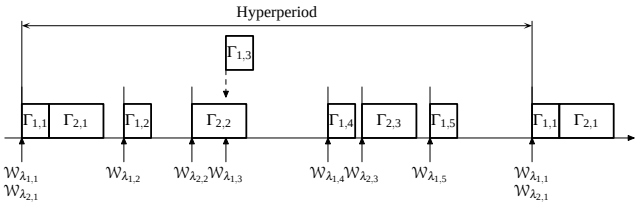


Figure 4. Ground jobs and non-ground jobs in a dynamic-priority system

In this example, there are seven ground jobs $\Gamma_{1,1}$, $\Gamma_{2,1}$, $\Gamma_{1,2}$, $\Gamma_{2,2}$, $\Gamma_{1,4}$, $\Gamma_{2,3}$, and $\Gamma_{1,5}$, and one non-ground job $\Gamma_{1,3}$, as shown in Figure 4. If the stationary system-level backlog PF $f_{\mathcal{B}_k^0}(\cdot)$ observed at the beginning of the hyper-period is given, the stationary job-level backlog PFs of all the ground jobs can be calculated by applying convolution and shrinking to $f_{\mathcal{B}_k^0}(\cdot)$. That is, $f_{\mathcal{V}_{1,1}}(w) = f_{\mathcal{W}_{\lambda_{1,1}}}(w) = f_{\mathcal{B}_k^0}(w)$, $f_{\mathcal{V}_{2,1}}(w) = f_{\mathcal{W}_{\lambda_{2,1}}}(w) = (f_{\mathcal{C}_{1,1}} \otimes f_{\mathcal{V}_{1,1}})(w)$, and so on.

However, for the non-ground job $\Gamma_{1,3}$, the stationary job-level backlog PF $f_{\mathcal{V}_{1,3}}(\cdot)$ is not equal to the stationary system-level backlog PF $f_{\mathcal{W}_{\lambda_{1,3}}}(\cdot)$ observed at its release time $\lambda_{1,3}$. The stationary system-level backlog PF $f_{\mathcal{W}_{\lambda_{1,3}}}(\cdot)$ includes the contribution from a lower-priority job $\Gamma_{2,2}$ that must be excluded for the job-level backlog PF $f_{\mathcal{V}_{1,3}}(\cdot)$. In this case, we observe that the job-level backlog PF $f_{\mathcal{V}_{1,3}}(\cdot)$ of the non-ground job $\Gamma_{1,3}$ can be calculated from the system-level (or job-level) backlog PF of the ground job $\Gamma_{1,2}$ that precedes $\Gamma_{1,3}$ and has a higher priority than $\Gamma_{1,3}$ (i.e., $\Gamma_{1,2}$ is the base ground job of the non-ground job $\Gamma_{1,3}$). It can be seen that the job-level backlog $\mathcal{V}_{1,3}$ of $\Gamma_{1,3}$ depends only on the system-level backlog $\mathcal{W}_{\lambda_{1,2}}$ of $\Gamma_{1,2}$ and the execution time of $\Gamma_{1,2}$. Thus, we calculate the stationary job-level backlog PF $f_{\mathcal{V}_{1,3}}(\cdot)$ of the non-ground job $\Gamma_{1,3}$ from that of its base ground job $\Gamma_{1,2}$ by applying convolution and shrinking to the system-level backlog

PF $f_{\mathcal{W}_{\lambda_{1,2}}}(\cdot)$ of the base ground job $\Gamma_{1,2}$ while ignoring the lower-priority ground job $\Gamma_{2,2}$. This approach can be generalized to compute the stationary job-level backlog PF of an arbitrary non-ground job $\Gamma_{i,j}$ as follows.

Theorem 1. *For any non-ground job $\Gamma_{i,j}$, the stationary job-level backlog PF $f_{\mathcal{V}_{i,j}}(\cdot)$ can be computed from the stationary system-level backlog PF $f_{\mathcal{W}_{\lambda_{k,l}}}(\cdot)$ of its base ground job, (i.e., the last ground job $\Gamma_{k,l}$ that precedes $\Gamma_{i,j}$ and has a priority higher than that of $\Gamma_{i,j}$) by iteratively applying convolution and shrinking to $f_{\mathcal{W}_{\lambda_{k,l}}}(\cdot)$ only for all the non-ground jobs that are released in the time interval $(\lambda_{k,l}, \lambda_{i,j}]$ and have priorities higher than that of $\Gamma_{i,j}$.* □

Note that, in a system scheduled by EDF, we can find the base ground job for any non-ground job, since there always exists a ground job that has an earlier deadline than the non-ground job. For a general dynamic-priority system, we developed a systematic approach to finding the base ground job for any non-ground job and computed the job-level backlog PF of the non-ground job in a systematic way, but we do not include it here for the sake of brevity. For more information, refer to [8].

5. Experimental results

In this section, we give experimental results obtained by the proposed stochastic analysis. First, we compare the results obtained from the proposed analysis method (both the exact solution and the approximated solution based on the Markov matrix truncation) with those obtained by Stochastic Time Demand Analysis (STDA) [7, 6] using the example given in [6]. Secondly, we compare the results obtained using the truncation method with those obtained from simulations while varying the average system utilization and the number of tasks.

5.1. Comparison with STDA

To compare the proposed analysis method with STDA, we used the same task sets given in [6], which are shown in Table 3. All three task sets in Table 3 are assumed to be scheduled by RM and have the same task parameters except for the execution time PFs of tasks given by a uniform PF that ranges from $C_i^{\min}$ to $C_i^{\max}$ ($C_i^{\max} = 2 \times \overline{C_i} - C_i^{\min}$) for task τ_i. For the task sets, the simulation results and the analytical results given by STDA are copied from [6] and compared with the results given by the proposed stochastic analysis, using the exact solution and the approximation obtained by the Markov matrix truncation. The results in Table 4 show that there is a significant difference between the deadline miss probability given by STDA and the one obtained by the proposed method, which is almost identical to the simulation result. This results from the critical instant assumption made in STDA. The negative impact on the degradation of analysis accuracy increases with an increase in the (maximum) system utilization. In the case of task τ_2 in S_2, the deadline miss probability given by STDA is more than six times the one given by the proposed method.

S	τ_i	T_i	D_i	$C_i^{\min}$	$\overline{C_i}$	$C_i^{\max}$	U	$U^{\max}$
S_1	τ_1	300	300	72	100	128	0.708	0.997
	τ_2	400	400	72	150	228		
S_2	τ_1	300	300	50	100	150	0.708	1.125
	τ_2	400	400	50	150	250		
S_3	τ_1	300	300	1	100	199	0.708	1.411
	τ_2	400	400	1	150	299		

Table 3. Task sets in [6]

S	τ_i	simulation	STDA	Exact analysis	Approximation
S_1	τ_1	**0.000** $\pm$ 0.000	**0.000**	**0.000**	**0.000**
	τ_2	**0.047** $\pm$ 0.001	**0.141**	**0.047**	**0.047**
S_2	τ_1	**0.000** $\pm$ 0.000	**0.000**	**0.000**	**0.000**
	τ_2	**0.074** $\pm$ 0.002	**0.489**	**0.074**	**0.074**
S_3	τ_1	**0.000** $\pm$ 0.000	**0.000**	**0.000**	**0.000**
	τ_2	**0.192** $\pm$ 0.001	**0.608**	**0.192**	**0.191**

Table 4. Results for the task sets in Table 3

5.2. Sensitivity to the average system utilization and the number of tasks

To assess the sensitivity of analysis accuracy to the average system utilization, we performed experiments with task sets consisting of three and five tasks while varying the average system utilization from 0.70 to 0.96. For each experiment, we also considered both RM and EDF scheduling to investigate the effect of the scheduling policy on analysis accuracy. The results (which we do not detail here due to space limit), showed that the approximated solutions from the Markov matrix truncation is almost equal to the simulation results for most cases. In the case of the task sets consisting of three tasks, the error between the approximated solution and the simulation result of the deadline miss probability for a task ranges from 0 to 0.006 for RM, and from 0 to 0.008 for EDF. In the case of the task sets consisting of five tasks, the error ranges from 0 to 0.015 for RM, and from 0 to 0.028 for EDF. According to the results, the error increases as the average system utilization increases. For example, in the case where the task set consisting of five tasks are scheduled by EDF, the error was 0, 0.002, 0.016, 0.028 when the average system utilization was 0.7, 0.8, 0.9, 0.96, respectively. The inaccuracy is due to the Markov matrix truncation, which converts the Markov matrix with an infinite dimension to one with a finite dimension. When the stationary system-level backlog PF we are trying to obtain has a non-negligible tail distribution beyond the truncation point, the Markov matrix truncation inevitably introduces approximation errors in the analysis. To reduce these errors, it is necessary to preserve the tail distribution as much

as possible by choosing a larger truncation point at the cost of greater computational overheads. For more information, the reader is referred to [8].

6. Conclusions and future work

In this paper we have introduced a model and a conceptual framework which allows statistical analysis of the response time of periodic tasks in a general priority-driven real-time system. The model requires *a priori* knowledge of the probability function (PF) of the execution time required by each job. We have derived formulae for calculating the backlog PF at any time, and the response time PF of any task. The statistical information derived from the analysis can be combined with deadlines, thus obtaining the probability of deadline misses for any task in the system. If this probability is zero for a task, then its deadline is guaranteed. In this sense the statistical analysis subsumes the classical response-time analysis. The analysis also allows for systems with $U^{\max} > 1$, which would not be feasible using classical analysis. Whenever $\bar{U} < 1$, the long-term statistical behavior can be described by modeling the backlog process as a Markov chain.

The model still has some limitations which can be investigated. It does not allow for uncertainty in the release times of the jobs, and assumes all tasks to be independent, so it does not allow for blocking in shared resources. Furthermore, the assumed independence is also statistical, i.e. the execution time distribution of a task is assumed to be independent of the values observed for the other tasks, or for previous instances of the same task. This assumption is often violated in real-world cases. When the number of jobs in a hyperperiod is large, finding the stationary distribution analytically is too expensive (although possible), so some numerical methods have been provided as approximations. In future work, the error introduced for such methods should be studied, and the development of a numerical method that can bound the error is needed.

Acknowledgement The authors of this paper are pleased to thank Prof. Guillem Bernat and Prof. Jane Liu as well as the anonymous reviewers for the valuable comments and observations they provided.

References

[1] L. Abeni and G. Buttazzo. Stochastic Analysis of a Reservation Based System. In *Proc. of the 9th International Workshop on Parallel and Distributed Real-Time Systems*, Apr. 2001.

[2] A. K. Atlas and A. Bestavros. Statistical Rate Monotonic Scheduling. In *Proc. of the 19th IEEE Real-Time Systems Symposium*, pages 123–132, Dec. 1998.

[3] G. Bernat, A. Colin, and S. Petters. WCET Analysis of Probabilistic Hard Real-Time Systems. In *Proc. of the 23rd IEEE Real-Time Systems Symposium*, Dec. 2002.

[4] A. Burns. Preemptive Priority Based Scheduling: An Appropriate Engineering Approach. In S. H. Son, editor, *Advances in Real-Time Systems*. Prentice Hall, 1994.

[5] J. L. Díaz, J. M. López, and D. F. García. Probabilistic Analysis of the Response Time in a Real-Time System. In *Proc. of the 1st CARTS Workshop on Advanced Real-Time Technologies*, Aranjuez, Spain, Oct. 2002. Also available as Technical Report at http://www.atc.uniovi.es/research/PART01.pdf.

[6] M. K. Gardner. *Probabilistic Analysis and Scheduling of Critical Soft Real-Time Systems*. PhD thesis, University of Illinois, Urbana-Champaign, 1999.

[7] M. K. Gardner and J. W. Liu. Analyzing Stochastic Fixed-Priority Real-Time Systems. In *Proc. of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Mar. 1999.

[8] K. Kim, L. L. Bello, C.-G. Lee, S. L. Min, and O. Mirabella. An Accurate Stochastic Analysis of General Priority-Driven Real-Time Systems. Technical report, School of Computer Science and Engineering, Seoul National University, 2002. Also available at http://archi.snu.ac.kr/khkim/sapds02.ps.

[9] K. Kim, L. L. Bello, S. L. Min, and O. Mirabella. On Relaxing Task Isolation in Overrun Handling to Provide Probabilistic Guarantees to Soft Real-Time Tasks with Varying Execution Times. In *Proc. of the 14th Euromicro Conference on Real-Time Systems*, Jun. 2002.

[10] L. Kleinrock. *Queueing Systems*. John Wiley & Sons, 1975.

[11] J. P. Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *Proc. of the 11th IEEE Real-Time Systems Symposium*, pages 201–209, Dec. 1990.

[12] J. P. Lehoczky. Real-Time Queueing Theory. In *Proc. of the 17th IEEE Real-Time Systems Symposium*, pages 186–195, Dec. 1996.

[13] J. P. Lehoczky. Real-Time Queueing Network Theory. In *Proc. of the 18th IEEE Real-Time Systems Symposium*, pages 58–67, Dec. 1997.

[14] J. P. Lehoczky, L. Sha, and Y. Ding. The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In *Proc. of the 10th IEEE Real-Time Systems Symposium*, Dec. 1989.

[15] J. Leung and J. Whitehead. On the Complexity of Fixed Priority Scheduling of Periodic Real-Time Tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[16] L. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of ACM*, 20(1):46–61, 1973.

[17] A. K. Mok and D. Chen. A Multiframe Model for Real-Time Tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, Oct. 1997.

[18] 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 *Proc. of the Real-Time Technology and Applications Symposium*, pages 164–173, Chicago, Illinois, May 1995.

[19] K. Tindell, A. Burns, and A. J. Wellings. An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks. *Real-Time Systems*, 6:133–151, 1994.