An Exact Stochastic Analysis of Priority-Driven Periodic Real-Time Systems and Its Approximations

Kanghee Kim[†] José Luis Díaz[‡] Lucia Lo Bello[§] José María López[‡]
Chang-Gun Lee[¶] Sang Lyul Min[†]

Abstract

This paper describes a stochastic analysis framework which computes the response time distribution and the deadline miss probability of individual tasks, even for systems with a maximum utilization greater than one. The framework is uniformly applied to fixed-priority and dynamic-priority systems and can handle tasks with arbitrary relative deadlines and execution time distributions.

Keywords: C.3.d Real-time and embedded systems, D.4.1.e Scheduling, D.4.8.g Stochastic analysis, G.3.e Markov processes

I. Introduction

Schedulability analysis methods for hard real-time systems presented in the literature are typically based on the periodic task model [1] and on worst-case assumptions on execution times [1], [2], [3] to provide a deterministic guarantee that all the jobs of every task in the system meet their deadlines. Such a deterministic timing guarantee, mandatory for hard real-time systems, is not required by soft real-time applications, which can be satisfied with a probabilistic guarantee that the deadline miss ratio of a task is below a given threshold. As a result, for soft real-time applications, the assumption that every job of a task requires the worst-case execution time can be relaxed in order to improve the system utilization. This is also the case for probabilistic hard real-time systems [4] where a probabilistic guarantee close to 0% suffices.

[†] Kanghee Kim and Sang Lyul Min ({khkim,symin}@archi.snu.ac.kr), *School of Computer Science and Engineering, Seoul National University* (Seoul, 151-742 Korea). This work was 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.

[‡] José Luis Díaz and José María López ({jldiaz,chechu}@uniovi.es), *Departamento de Informática*, *Universidad de Oviedo* (33204, Gijón, Spain)

[§] Lucia Lo Bello ({llobello}@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.)

In this paper we propose and discuss a stochastic analysis framework which does not introduce any worst-case or restrictive assumptions into the analysis and is applicable to general priority-driven real-time systems. The framework builds upon Stochastic Time Demand Analysis (STDA) [5] as far as the techniques used to compute the response time distributions of tasks are concerned, but differs from STDA in several respects. First, our framework considers all possible execution scenarios in order to obtain the exact response time distributions of the tasks, while STDA focuses on particular execution scenarios starting at a critical instant. Second, while STDA addresses only fixed-priority systems such as Rate Monotonic [1] and Deadline Monotonic [6], our framework extends to dynamic-priority systems such as Earliest Deadline First [1]. Third, STDA only provides valid results if the maximum system utilization is less than or equal to 1, while our analysis is applicable to systems with a maximum utilization greater than 1.

In our framework, in order to consider all the possible execution scenarios in the system, we analyze a whole hyperperiod of the given task set, that is, a period having the same duration as the least common multiple of the periods of all the tasks. Moreover, to handle cases in which the maximum system utilization is greater than 1, and so one hyperperiod may affect the next one, we model the system as a Markov process over an infinite sequence of hyperperiods. This modeling leads us to solve an infinite number of linear equations. Here we present three different methods to solve the problem. One method gives the exact solution, while the others give approximated solutions. We compare these methods, in terms of accuracy and complexity, and discuss experimental results.

The rest of the paper is organized as follows. In Section II related work is outlined to establish the context of and motivation for our work. In Section III, the system model is explained. Sections IV and V describe the stochastic analysis framework including the exact and approximation methods. Section VI gives the experimental results obtained. Finally, Section VII concludes the paper with directions for future research.

II. Related work

Several studies have addressed the variability of task execution times in analyzing the schedulability of a given task set. Probabilistic Time Demand Analysis (PTDA) is a stochastic extension of Time Demand Analysis (TDA) [2] and can only deal with tasks with relative deadlines smaller than or equal to the periods. Stochastic Time Demand Analysis (STDA), on the other hand, is a stochastic extension of General Time Demand Analysis [3] and can handle tasks with relative deadlines greater than the periods. Like the original TDA, both methods assume the critical instant at which the task being analyzed and all the higher-priority tasks are released or arrive at the same time. Although this worst-case assumption simplifies the analysis, it only results in

an upper bound on the deadline miss probability, the conservativeness of which depends on the number of tasks and the average utilization of the system. Moreover, both analyses are valid only when the maximum utilization of the system does not exceed 1.

Other analysis methods based on simplifying worst-case assumptions are the one proposed by Manolache et al. [7], which addresses only uniprocessor systems, and the one by Leulseged and Nissanke [8], which extends to multiprocessor systems. These methods, like the one presented in this paper, cover general priority-driven systems including both fixed-priority and dynamic-priority systems. However, to limit the scope of the analysis to a single hyperperiod, both methods assume that the relative deadlines of tasks are shorter than or equal to their periods and that all the jobs that miss their deadlines are dropped. Moreover, in [7], all the tasks are assumed to be non-preemptible to simplify the analysis.

The Real-Time Queueing Theory (RTQT) [9] extends the classical queueing theory to real-time systems. RTQT is not limited to a particular scheduling algorithm and can be extended to real-time queueing networks. However, it is based on a restrictive assumption, i.e. the heavy traffic assumption (which means that the average system utilization is close to 1), so it is only applicable to systems where such an assumption holds. Moreover, it only considers one class of tasks such that the interarrival times and execution times are identically distributed.

Other stochastic analysis methods in the literature include the one proposed by Abeni and Buttazzo [10], and the method with Statistical Rate Monotonic Scheduling (SRMS) [11]. Both assume reservation-based scheduling algorithms so that the analysis can be performed as if each task had a dedicated (virtual) processor. Each task is provided with a guaranteed budget of processor time in every period [10] 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) [11]. So, the deadline miss probability of a task can be analyzed independently of the other tasks, assuming the guaranteed budget. However, these stochastic analysis methods are not applicable to general priority-driven systems due to the modification of the original priority-driven scheduling rules or the use of reservation-based scheduling algorithms.

III. System model

We assume a uniprocessor system that consists of a set of n independent periodic tasks $S = \{\tau_1, \ldots, \tau_n\}$, each task τ_i ($1 \leq i \leq n$) being modeled by the tuple $(T_i, \Phi_i, \mathcal{C}_i, D_i)$, where T_i is the task period, Φ_i the initial phase, $\mathcal{C}_i$ the execution time, and D_i the relative deadline. The execution time is a discrete random variable[1] with a given probability mass function (PMF), denoted by $f_{\mathcal{C}_i}(\cdot)$, where $f_{\mathcal{C}_i}(c) = \mathbb{P}\{\mathcal{C}_i{=}c\}$. Without loss of generality, the phase Φ_i of each

[1]Throughout this paper, we use a calligraphic typeface to denote random variables, e.g., $\mathcal{C}$, $\mathcal{W}$, and $\mathcal{R}$.

task τ_i is assumed to be smaller than T_i. The relative deadline D_i can be smaller than, equal to, or greater than T_i.

The system utilization is defined as the sum of the utilizations of all the tasks. Due to the variability of task execution times, here three system utilizations are defined, i.e. the minimum $U^{\min}$, the maximum $U^{\max}$ and the average system utilization $\bar{U}$, which are calculated using the minimum, maximum and average task execution times, respectively. In addition, a hyperperiod of the task set is defined as a period of length $T_H = \mathrm{lcm}_{1 \le i \le n}\{T_i\}$.

Each task consists of an infinite sequence of jobs, whose release times are deterministic. If we denote the j-th job of task τ_i by $J_{i,j}$, its release time $\lambda_{i,j}$ is equal to $\Phi_i + (j-1)T_i$. Each job $J_{i,j}$ requires an execution time, which is described by a random variable following the given PMF $f_{\mathcal{C}_i}(\cdot)$ of the task τ_i, and is assumed to be independent of other jobs of the same task and those of other tasks. However, throughout the paper we use a single index j for the job subscript, since the task that the job belongs to is not important in describing our analysis framework.

The scheduling model we assume is a general, preemptive, priority-driven one that covers both fixed-priority systems such as Rate Monotonic (RM) and Deadline Monotonic (DM), and dynamic-priority systems such as Earliest Deadline First (EDF). The only limitation is that once a priority is assigned to a job, it never changes, which is called a job-level fixed-priority model [12]. We denote the priority of job J_j by a priority value p_j. A higher priority value means a lower priority. At any time, the job with the highest priority is always served first. If two or more jobs with the same priority are ready at the same time, they are scheduled according to the FCFS (First Come First Served) rule.

The response time of a job J_j is a random variable, $\mathcal{R}_j$, the PMF of which has to be obtained by analysis. D_i is the task relative deadline, and M_i the maximum allowable probability of missing it. Task τ_i is said to be schedulable if $\mathbb{P}\{\mathcal{R}_i > D_i\} \le M_i$.

IV. Stochastic Analysis Framework

A. Overview

What follows is a brief summary of the proposed analysis method, based on an example. For details and mathematical proofs, the reader is referred to [13] and [14].

The response time of a job J_j is given by $\mathcal{R}_j = \mathcal{W}_{p_j}(\lambda_j) + \mathcal{C}_j + \mathcal{I}_j$, where $\mathcal{W}_{p_j}(\lambda_j)$ is the backlog of priority p_j at time λ_j, which represents the workload of jobs with priority p_j and higher that have not yet been processed just before the release time λ_j of J_j. $\mathcal{C}_j$ is the execution time of job J_j. $\mathcal{I}_j$ is the interference on J_j of all the jobs with a higher priority than job J_j, released after job J_j. Note that all the terms in the equation are random variables. This is the

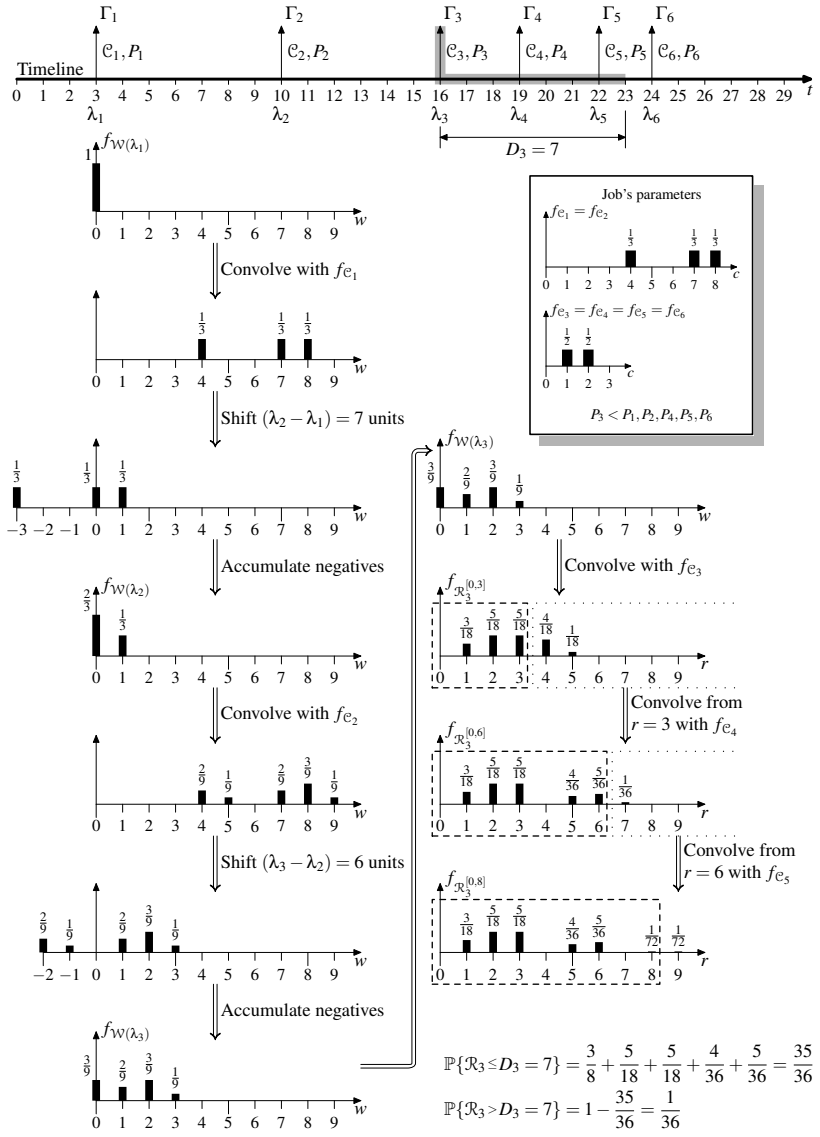


Fig. 1. Example of calculation of response time distribution for a job

stochastic counterpart of a well-known deterministic equation that provides the response time of a job under a preemptive priority-driven scheduling policy (see eq. (16) in [15]).

None of the jobs with a priority less than p_j released before λ_j has any influence on the response time of job J_j. In addition, none of the jobs with a priority p_j or less released after λ_j has any influence on the response time of job J_j. To simplify the notation, we assume that all these jobs are removed in the calculation process of $\mathcal{R}_j$, and the task indices updated accordingly. In addition, we can remove the subindex p_j from $\mathcal{W}_{p_j}$, since all jobs considered have a priority p_j or higher.

Figure 1 illustrates an example in which the response time of job J_3 is computed, following the algorithm described in [13]. The calculation starts with null backlog at the release time of the first job, J_1, i.e., $\mathcal{W}(\lambda_1)=0$. Note that, in general $\mathcal{W}(\lambda_1)$ is a random variable, but in this case its value is zero, since we assume that the system begins with J_1.

The backlog distribution at λ_2, denoted by $\mathcal{W}(\lambda_2)$, is calculated by convolving $f_{\mathcal{W}(\lambda_1)}$ with

$f_{\mathcal{C}_1}$, shifting the result by $(\lambda_2 - \lambda_1)$ time units to the left and accumulating the probability values in the negative range onto the origin (see fig.1). In the same way, the backlog distribution at λ_3, $\mathcal{W}(\lambda_3)$, can be calculated by convolving $f_{\mathcal{W}(\lambda_2)}$ with $f_{\mathcal{C}_2}$, shifting the result by $(\lambda_3 - \lambda_2)$ time units to the left and accumulating the probability values in the negative range onto the origin.

Let us define the function $\text{SHRINK}(\mathcal{W}, \Delta)$, which produces a new random variable whose PMF is equal to the PMF of $\mathcal{W}$, left-shifted by the amount Δ and with all values for negative abscissae accumulated at the origin. That is:

$$f_{\text{SHRINK}(\mathcal{W},\Delta)}(x) = \begin{cases} 0 & \text{if } x < 0 \\ \sum_{w=-\infty}^{0} f_{\mathcal{W}}(w+\Delta) & \text{if } x = 0 \\ f_{\mathcal{W}}(x+\Delta) & \text{if } x > 0 \end{cases} \tag{1}$$

Using this function, the calculation of $\mathcal{W}(\lambda_j)$ can be expressed as

$$\begin{aligned} &\mathcal{W}(\lambda_1) = 0 \\ &\mathcal{W}(\lambda_j) = \text{SHRINK}(\mathcal{W}(\lambda_{j-1}) + \mathcal{C}_{j-1}, \lambda_j - \lambda_{j-1}) \quad \text{for } j > 1 \end{aligned} \tag{2}$$

Note that the distribution of a sum of random variables is obtained by convolving their distributions. Once $\mathcal{W}(\lambda_j)$ has been calculated, in order to calculate $\mathcal{R}_j$ it is necessary to add the execution time $\mathcal{C}_j$ and the interference of jobs released later than λ_j. Figure 1 depicts the process.

The distribution $f_{\mathcal{W}(\lambda_3)}$ is convolved with $f_{\mathcal{C}_3}$. The resulting distribution, denoted $\mathcal{R}_3^{[0,\lambda_4-\lambda_3]}$, is a random variable that describes the response time distribution of job J_3 assuming that all the interfering jobs J_4, J_5, and J_6 do not exist. The actual distribution of $\mathcal{R}_3$ in the range $[0, \lambda_4 - \lambda_3]$ coincides with that of $\mathcal{R}_3^{[0,\lambda_4-\lambda_3]}$ in the same range. The complete actual distribution of $\mathcal{R}_3$ is computed below by considering each interfering job.

The distribution $f_{\mathcal{R}_3^{[0,\lambda_4-\lambda_3]}}$ is convolved from[2] $r = (\lambda_4 - \lambda_3)$ with $f_{\mathcal{C}_4}$. The resulting distribution, whose random variable is denoted by $\mathcal{R}_3^{[0,\lambda_5-\lambda_3]}$, is the response time distribution of job J_3 assuming that the interfering jobs J_5 and J_6 do not exist. The actual response time distribution of $\mathcal{R}_3$ in the range $[0, \lambda_5 - \lambda_3]$ coincides with that of $\mathcal{R}_3^{[0,\lambda_5-\lambda_3]}$ in the same range.

The iteration process continues until the relative deadline of J_j is included in the interval of one of the random variables. In the example of Figure 1, the iteration ends with the calculation of $\mathcal{R}_3^{[0,\lambda_6-\lambda_3]}$ for the relative deadline of value 7 for J_3. At that moment, the probability of J_3 meeting its deadline can be computed (of value $35/36$), and therefore the probability of missing its deadline (of value $1/36$).

[2]The "convolve from" operation is formally defined in eq. (3)

Let us define the function $\mathrm{CF}(\mathcal{R},\Delta,\mathcal{C})$, which convolves the tail distribution of $\mathcal{R}$ defined in the range $[\Delta,\infty)$ (i.e., $\mathcal{R}^{[\Delta,\infty)}$) and $\mathcal{C}$. The result is a new random variable, whose distribution is obtained as follows:

$$f_{\mathrm{CF}(\mathcal{R},\Delta,\mathcal{C})}(x) = \begin{cases} f_{\mathcal{R}}(x) & \text{for } x \le \Delta \\ \sum_{i=\Delta+1}^{\infty} f_{\mathcal{R}}(i) \cdot f_{\mathcal{C}}(x-i) & \text{for } x > \Delta \end{cases} \tag{3}$$

Using this operator, the calculation of $\mathcal{R}_j$ with interfering jobs can be expressed as

$$\begin{aligned} \mathcal{R}_j^{[0,\lambda_{j+1}-\lambda_j]} &= \mathcal{W}(\lambda_j) + \mathcal{C}_j \\ \mathcal{R}_j^{[0,\lambda_{k+1}-\lambda_j]} &= \mathrm{CF}(\mathcal{R}_j^{[0,\lambda_k-\lambda_j]}, \lambda_k - \lambda_j, \mathcal{C}_k) \quad \text{for } k > j \end{aligned} \tag{4}$$

The iteration can stop when $\lambda_{k+1} - \lambda_j \ge D_j$.

In theory, the probability of a task missing its deadline is calculated by averaging the probabilities of all its jobs missing that deadline, but in practice the number of these jobs is infinite. It can be proved that, when $\bar{U} < 1$ the system becomes stable and in the steady state, the probability of a job missing its deadline becomes constant for the same job released one, two or any number of hyperperiods later. Thus, it would suffice to compute the steady-state response time distribution for all the jobs released in a single hyperperiod to obtain the exact response time distribution for any task. However, in order to compute the steady-state response time of any job J_j, it is necessary to compute the steady-state backlog of priority p_j at its release instant λ_j. This computation can be optimized based on the observation that there exists a dependency among the backlogs of the different jobs. Once the steady-state backlog distribution has been computed for a job, this result can be reused to compute the steady-state backlog for all the other jobs in the steady-state hyperperiod.

B. Dependency among the backlogs

To show that there exist dependencies between the backlogs, we first classify all the jobs in a hyperperiod into *ground jobs* and *non-ground jobs*. A ground job is defined as a job that has a lower priority than those of all the jobs previously released. That is, J_j is a ground job if and only if $p_k \le p_j$ for all jobs J_k such that $\lambda_k < \lambda_j$. A non-ground job is a job that is not a ground job. One important implication from the ground job definition is that the p_j-backlog of a ground job is always equal to the total backlog in the system observed at its release time. We call the total backlog *system backlog* and denote it by $\mathcal{W}(t)$, i.e., without the subscript p_j denoting the priority level. So, for a ground job J_j, $\mathcal{W}_{p_j}(\lambda_j) = \mathcal{W}(\lambda_j)$.

We can capture backlog dependencies between the ground and non-ground jobs. For each non-ground job J_j, we search for the last ground job that is released before J_j and has a priority higher than or equal to that of J_j. Such a ground job is called the *base job* for the non-ground

job. From this relation, we can observe that the p_j-backlog of the non-ground job J_j directly depends on that of the base job. Note that such backlog dependencies exist even between ground jobs, and can still be captured under the concept of the base job. As a result, all the backlog dependencies among the jobs can be depicted with a tree (which can degenerate to a list, in some cases). Let us particularize this scheme for the case of RM and EDF scheduling.

For RM scheduling, the ground jobs are those belonging to the task with the lowest priority. The backlog of each ground job can be computed from the backlog of the preceding ground job. The dependency "tree" is in this case a simple list (or chain) which connects all ground jobs (i.e., all lowest-priority jobs). However, the non-ground jobs are not connected to this list, because they do not have a base job (all jobs released before any non-ground job have a priority lower than that of the non-ground job). This means that the backlog of the non-ground job cannot be derived from the backlog of the ground jobs. This problem can be addressed in the following way; once the ground jobs have been solved, the lowest-priority task is removed from the system model, and the resulting system is analyzed again. In this new system there will be the new lowest-priority task (i.e., the second lowest-priority task in the original system model), which will generate a new set of ground jobs, to which the same methodology is applied. Therefore, under RM we have n separated dependency lists, n being the number of priority levels. The steady-state backlog has to be computed for the job at the head of each list, which is the first lowest-priority job in the hyperperiod, using the methods described in the next section. Then, the backlog of the remaining jobs in the list is computed using eq. (1), using as the initial backlog the steady-state backlog computed for the head of the list.

For EDF scheduling, it can be shown [14] that a ground job exists in each hyperperiod, and that all non-ground jobs have a base job within a finite time window (the size of this window is $\max\{D_i\} + T_H$, $\max\{D_i\}$ being the maximum among the deadlines for all the tasks). Therefore, all the jobs are linked in a single dependency tree, whose root is the first ground job in the hyperperiod. Once the steady-state backlog has been found using one of the methods in the next section for this ground job, the steady-state backlog of all the other jobs can be computed by traversing this dependency tree while applying eq. (1).

Therefore, the only difference between dynamic-priority systems and fixed-priority systems is that for the former the backlog distributions of all the jobs are computed at once with the single backlog dependency tree, while for the latter they are computed by iterative analysis over the n priority levels, which results in n backlog dependency lists. In any case, we need a method to determine the steady-state backlog of the root job in the tree (or the head job in each list). This method is presented in the next section.

V. Steady-state Backlog Analysis

Let us assume an infinite sequence of hyperperiods, the first of which starts at the release time λ_j of the ground job being considered, J_j. Let $f_{\mathcal{B}_k}$ be the distribution of the system backlog $\mathcal{B}_k$ observed at the release time of the ground job in the hyperperiod k, that is, at the instant $\lambda_j + kT_H$. It can be proved that, if the average system utilization $\bar{U}$ is less than 1, there exists a stationary (or limiting) distribution $f_{\mathcal{B}_\infty}$ of the system backlog $\mathcal{B}_k$ such that

$$\lim_{k\to\infty} f_{\mathcal{B}_k} = f_{\mathcal{B}_\infty}$$

For the special case where $U^{\max}$ is also less than 1, the system backlog distributions $f_{\mathcal{B}_k}$ of all the hyperperiods are identical. That is, $f_{\mathcal{B}_1} = \cdots = f_{\mathcal{B}_k} = \cdots = f_{\mathcal{B}_\infty}$. In this case, the stationary backlog distribution $f_{\mathcal{B}_\infty}$ can easily be computed by considering only the finite sequence of jobs released before the instant λ_j. That is, we simply have to apply eq. (1) along the finite sequence of jobs released in $[0, \lambda_j)$, while assuming that the system backlog at time 0 is 0.

For the case where $U^{\max} > 1$ (while $\bar{U} < 1$), the system backlog distributions are different in each hyperperiod, but they converge towards a *limiting* distribution $f_{\mathcal{B}_\infty}$. This distribution can be approximated by simple iteration of eq. (1) along several hyperperiods, until two successive distributions are close enough, or it can be computed exactly by the Markovian analysis presented in the next subsection.

A. Exact solution

For a general case where $U^{\max} > 1$, in order to compute the exact solution for the stationary backlog distribution $f_{\mathcal{B}_\infty}$, we show that the stochastic process defined with the sequence of random variables $\{\mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_k, \ldots\}$ is a Markov chain. To do this, let us express the PMF of $\mathcal{B}_k$ in terms of the PMF of $\mathcal{B}_{k-1}$ using the concept of conditional probabilities.

$$\mathbb{P}\{\mathcal{B}_k{=}x\} = \sum_y \mathbb{P}\{\mathcal{B}_{k-1}{=}y\}\mathbb{P}\{\mathcal{B}_k{=}x \mid \mathcal{B}_{k-1}{=}y\} \tag{5}$$

We can see that the conditional probabilities $\mathbb{P}\{\mathcal{B}_k{=}x \mid \mathcal{B}_{k-1}{=}y\}$ do not depend on k, since all hyperperiods receive the same sequence of jobs with the same execution time distributions. That is, $\mathbb{P}\{\mathcal{B}_k{=}x \mid \mathcal{B}_{k-1}{=}y\} = \mathbb{P}\{\mathcal{B}_1{=}x \mid \mathcal{B}_0{=}y\}$. This leads us to the fact that the PMF of $\mathcal{B}_k$ depends only on that of $\mathcal{B}_{k-1}$, and not on those of $\{\mathcal{B}_{k-2}, \mathcal{B}_{k-3}, \ldots\}$. The stochastic process is thus a Markov chain. We can rewrite Equation (5) in matrix form as follows

$$\mathbf{b}_k = \mathbf{P}\,\mathbf{b}_{k-1} \tag{6}$$

where $\mathbf{b}_k$ is a column vector $\left[\mathbb{P}\{\mathcal{B}_k{=}0\}, \mathbb{P}\{\mathcal{B}_k{=}1\}, \ldots\right]^\intercal$, i.e., the PMF of $\mathcal{B}_k$, and $\mathbf{P}$ is the Markov matrix which consists of the transition probabilities $\mathbf{P}(x,y)$ defined as

$$\mathbf{P}(x,y) = \mathbb{P}\{\mathcal{B}_k{=}x \mid \mathcal{B}_{k-1}{=}y\} = \mathbb{P}\{\mathcal{B}_1{=}x \mid \mathcal{B}_0{=}y\}.$$

Thus, the problem of computing the exact solution $\boldsymbol{\pi}$ for the stationary backlog distribution, i.e., $\left[\mathbb{P}\{\mathcal{B}_\infty{=}0\},\mathbb{P}\{\mathcal{B}_\infty{=}1\},\ldots\right]^\intercal$, is equivalent to solving the equilibrium equation $\boldsymbol{\pi}=\mathbf{P}\boldsymbol{\pi}$. It can be shown that, whenever $\bar{U}<1$, the solution to this equation is unique.

Theoretically, when $k\to\infty$, the system backlog can be arbitrarily long, since $U^{\max}>1$. This means that the Markov matrix has an infinite size, and therefore, trying to find $\boldsymbol{\pi}$ leads to an infinite set of linear equations, which cannot be solved. However, matrix $\mathbf{P}$ has a regular structure which simplifies the problem. For any system, it can be shown that there exists an integer r, such that, for any $j>r$, the coefficients of column j in $\mathbf{P}$ are the same as the coefficients of column $j-1$, but shifted one position down. This regular structure means that the infinite set of linear equations can be reduced to a finite set of linear equations plus a recurrence relation. From these equations, the exact solution $\boldsymbol{\pi}$ can be found. The solution is of infinite length, but an expression which gives each of its coefficients in closed form can be obtained for any system. The reader can find more details in [13].

B. Approximations

The exact solution is interesting from a theoretical point of view. However, the practical implementation of the exact method is not exempt from problems. First, the computational cost of the method is very high. Second, the nature of the real numbers involved in the calculations leads to numerical problems when implemented in a computer. The exact method requires the matrix $\mathbf{P}$ to be computed up to column r in which the repetitive structure appears. Each column of $\mathbf{P}$ requires eq. (1) to be applied through all the jobs in one hyperperiod. The coefficients in column r give a recurrence relation, whose solution involves the diagonalization of a matrix. The size of this matrix depends on the number of non-null coefficients in column r of $\mathbf{P}$. In practice, the matrix to diagonalize will be huge. Moreover, this matrix has a large number of coefficients very close to zero, while other coefficients have very large values, so it is ill-conditioned for diagonalization.

Due to these problems, computation of the exact solution is only possible for systems with a small number of tasks, and other approximated methods should be investigated. One possible approximation is to truncate the original Markov matrix $\mathbf{P}$, creating a new finite matrix $\mathbf{P}'$. In this way, the equation $\boldsymbol{\pi}'=\mathbf{P}'\boldsymbol{\pi}'$ leads to a finite set of linear equations which can easily be solved on a computer. The solution $\boldsymbol{\pi}'$ will be an approximation of the exact solution $\boldsymbol{\pi}$. The accuracy of the approximation depends on the truncation point for $\mathbf{P}$, an issue that requires further investigation.

Another approximation is the iterative method already mentioned. This method has the advantage that computation of the Markov matrix $\mathbf{P}$ is not necessary, and thus it is computationally

task set		T_i	execution times			utilizations			RM					EDF			
			$C_i^{\min}$	$\bar{C}_i$	$C_i^{\max}$	$U^{\min}$	$\bar{U}$	$U^{\max}$	simulation	STDA	exact	trunc	iterat.	simulation	exact	trunc	iterat.
A	τ_1	20	4	6	10	.58	.82	1.27	.0000 ± .0000	.0000	.0000			.0001 ± .0000	.0001	.0001	.0001
	τ_2	60	12	16	22				.0000 ± .0000	.0000	.0000			.0000 ± .0000	.0000	.0000	.0000
	τ_3	90	16	23	36				.0940 ± .0025	.3931	.0940	.0940	.0940	.0000 ± .0000	.0000	.0000	.0000
B	τ_1	20	4	6	10	.58	.87	1.27	.0000 ± .0000	.0000	.0000			.0013 ± .0002	.0013	.0013	.0013
	τ_2	60	12	17	22				.0000 ± .0000	.0000	.0000			.0005 ± .0002	.0005	.0005	.0005
	τ_3	90	16	26	36				.2173 ± .0033	.6913	.2170	.2170	.2170	.0000 ± .0001	.0000	.0000	.0000
C	τ_1	20	4	7	10	.58	.92	1.27	.0000 ± .0000	.0000	.0000			.0223 ± .0013	.0224	.0224	.0224
	τ_2	60	12	17	22				.0000 ± .0000	.0000	.0000			.0168 ± .0014	.0169	.0169	.0169
	τ_3	90	16	26	36				.3849 ± .0052	.9075	.3852	.3852	.3852	.0081 ± .0011	.0081	.0081	.0081
C1	τ_1	20	3	7	11	.46	.92	1.38	.0000 ± .0000	.0000	.0000			.0626 ± .0031	.0630	.0627	.0627
	τ_2	60	10	17	24				.0000 ± .0000	.0000	.0000			.0604 ± .0038	.0610	.0607	.0607
	τ_3	90	13	26	39				.4332 ± .0065	.9209	.4334	.4334	.4334	.0461 ± .0032	.0466	.0463	.0463
C2	τ_1	20	2	7	12	.34	.92	1.50	.0000 ± .0000	.0000	.0000			.1248 ± .0058	N.A.	.1250	.1250
	τ_2	60	8	17	26				.0002 ± .0001	.0018	.0002	.0002	.0002	.1293 ± .0064		.1296	.1296
	τ_3	90	10	26	42				.4859 ± .0081	.9339	N.A.	.4860	.4860	.1136 ± .0063		.1138	.1138

TABLE I

TASK SETS USED IN THE EXPERIMENTS AND RESULTS OF THE DIFFERENT ANALYSIS METHODS

more efficient. The accuracy of this approximation can be controlled by comparing the solution $\boldsymbol{\pi}'$ obtained in each iteration with the one obtained in the previous iteration. The number of iterations required for a given accuracy level is unknown in advance.

VI. EXPERIMENTAL RESULTS

This section presents experimental results obtained using our analysis framework. We assess the complexity and accuracy of all the methods proposed here, also comparing their results with those obtained by STDA [5]. Secondly, we evaluate the complexity of the backlog and interference analysis by experiments. We investigate the effect on the backlog and interference analysis of various n (the number of jobs), m (the maximum length of the execution time distributions), $\bar{T}$ (the average interarrival time), and k (the degree of interference).

A. Comparison between the solution methods

We use the task sets shown in Table I, which consist of three tasks with the same periods, deadlines (equal to the periods), and null phases, which result in the same backlog dependency tree for a given scheduling algorithm. The task sets only differ in their execution time distributions. For task sets A, B, and C, the minimum and maximum execution times for each task do not change, while the average execution time is varied. This allows us to evaluate the effect of the average system utilization $\bar{U}$ on the stationary backlog distribution. For task sets C, C1, and C2, the average execution time of each task is fixed, while the whole execution time distribution is gradually stretched. This allows us to evaluate the effect of the maximum system utilization $U^{\max}$ on the stationary backlog distribution, while fixing the average system utilization $\bar{U}$.

Table I also summarizes the results of our stochastic analysis and, for the RM case, the results obtained by STDA. The deadline miss probability (DMP) for each task obtained from

task set	SSBD computation time (seconds)						
	exact	trunc			iterative		
		δ=10^{-3}	δ=10^{-6}	δ=10^{-9}	δ=10^{-3}	δ=10^{-6}	δ=10^{-9}
A	.13	.00	.00	.00	.00	.00	.00
		p=2	p=15	p=25	I=2	I=2	I=3
B	.13	.00	.00	.01	.00	.00	.01
		p=8	p=23	p=37	I=2	I=3	I=6
C	.15	.01	.03	.07	.00	.01	.03
		p=29	p=63	p=96	I=4	I=12	I=20
C1	.31	.02	.10	.25	.01	.05	.21
		p=54	p=115	p=173	I=7	I=20	I=35
C2	N.A.	.07	.31	.82	.02	.23	.88
		p=86	p=181	p=272	I=10	I=30	I=52

TABLE II

ANALYSIS TIME COMPARISON BETWEEN THE SOLUTION METHODS

the stationary system backlog distribution computed by each method (i.e., exact, Markov matrix truncation, iterative) and the average deadline miss ratio (DMR) and standard deviation obtained from simulations are shown. For the truncation and iterative methods, the values of the control parameters p (the size of the truncated matrix $\mathbf{P}'$) and I (the number of hyperperiod iterations) are shown in Table II. The average DMR is obtained by averaging the deadline miss ratios measured from 100 simulation runs of each task set, performed during 5000 hyperperiods. To implement the exact method and the Markov matrix truncation method, we used the Intel linear algebra package called Math Kernel Library 5.2.

Table I shows that our analysis results are almost identical to the simulation results, regardless of the solution method used. For the RM case, we can observe significant differences between the DMPs given by STDA and those obtained by our analysis. For example, for task τ_3 in task set A, the DMP given by STDA (39.3%) is more than four times that given by our analysis (9.4%). As $\bar{U}$ or $U^{\max}$ increases, the DMP computed by STDA gets even worse. This results from the critical instant assumption made in STDA. Our implementation of the exact method could not provide a numerically valid result for task set C2 (in the RM case, only for task τ_3) due to the limited precision of the numerical package we used. For the same reason, a small difference is observed between the DMP computed by the exact method and those computed by the approximation methods for task set C1, scheduled by EDF. However, this precision problem can be overcome simply by using a numerical package with greater precision.

Table II shows the analysis time required in the EDF case [3] by each solution method to compute the stationary system backlog distributions. This time does not include the time taken by the backlog dependency tree generation, which is almost 0, and the time required by the backlog and interference analysis, which is less than 10 ms. The table also shows the values of the control parameters p and I for the truncation and iterative methods. For fair comparison between the

[3]The analysis time was measured with a Unix system call called **times()** on a personal computer equipped with a Pentium Processor IV 2.0 GHz and 256 MB main memory.

two approximation methods, we define an accuracy level $\delta = ||SSBD_{exact} - SSBD_{approx}||$, where $SSBD_{exact}$ is the exact solution of the stationary system backlog distribution and $SSBD_{approx}$ is the approximated solution computed by either of the methods.

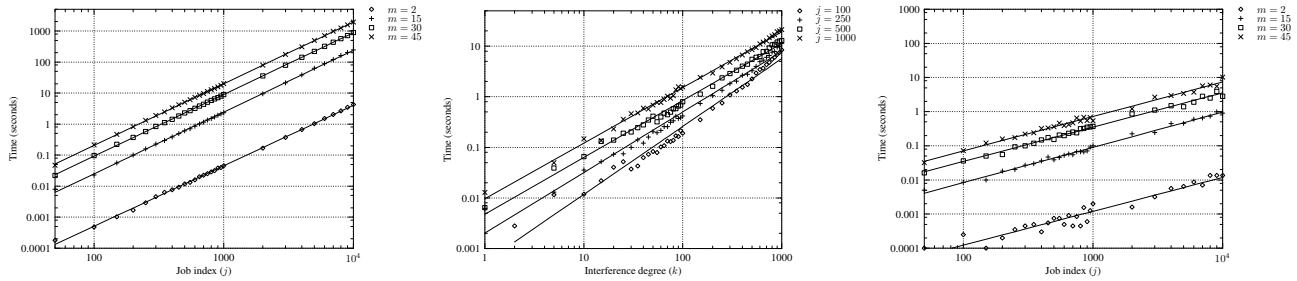
Table II shows that both the SSBD computation time and the associated control parameters used to obtain solutions with the required accuracy levels $\delta = 10^{-3}, 10^{-6}, 10^{-9}$ (The DMPs shown in Table I for the truncation and iterative methods were obtained at an accuracy level of $\delta = 10^{-6}$). For task sets A to C, as $\bar{U}$ increases, the analysis time rapidly increases for the truncation and iterative methods, while it remains almost constant for the exact one. The reason for this is that as $\bar{U}$ increases, the probability values of the stationary backlog distribution spread more widely, so both approximation methods have to compute the solution for a wider backlog range. Both methods should therefore use a larger value for the p and I parameters in order to achieve the required accuracy level. This spread of the stationary probability values does not affect the analysis time for the exact method, as most of this time is spent in solving the linear system, the size of which only depends on $U^{\max}$ and not on $\bar{U}$. The above observation also holds for the case of task sets C to C2 where, due to the increasing $U^{\max}$, the SSBD spreads even more widely. The analysis time taken by the exact method also increases, as the size of the resulting regular structure inherent to the Markov matrix becomes large due to the increasing length of the execution time distributions.

To summarize, if $\bar{U}$ and/or $U^{\max}$ is significantly high, the approximation methods require a long computation time for high accuracy, possibly longer than that of the exact method. However, if $\bar{U}$ is not close to 1, e.g., less than 0.8, the methods can provide highly accurate solutions at a considerably lower complexity.

B. Complexity evaluation of the backlog and interference analysis

We generated synthetic systems, varying the system parameters n, m, and $\bar{T}$, while fixing $\bar{U}$. Each system comprises n jobs with the same execution time distribution of length m and mean interarrival time $\bar{T}$. The shapes of the distributions of the job execution time and the interarrival time are determined in such a way that the fixed average system utilization is maintained, even if they have no influence on the complexity of the backlog and interference analysis[4]. For each system generated, we perform backlog and interference analysis, assuming a null backlog at the beginning of the analysis. For each of the n jobs, we measure the time taken by backlog analysis and interference analysis separately. In this measurement, the backlog analysis time for the j-th job is defined as the time taken to apply the convolve-shrink procedure from the first job J_1 (with the null backlog) to job J_j.

[4]The backlog and interference analysis time is not affected by the actual values of the probabilities composing the distributions.



(a) Backlog analysis time, for $\bar{T} = m/2$ ($U^{\max} \approx 2$)

(b) Interference analysis time, varying k ($m = 20$ and $\bar{T} = 1$)

(c) Interference analysis time, varying m ($k = 10$ and $\bar{T} = 2$)

Fig. 2. Analysis time

Figure 2(a) shows the backlog analysis time (in seconds) measured for each job J_j, while varying m and $\bar{T}$. (Note that both the x-axis and the y-axis are in logarithmic scale.) The figure shows that the backlog analysis time for each job increases in polynomial order $O(j^2m^2)$ [14]. However, due to the backlog dependencies, the backlog analysis for the j-th job may be efficiently performed in a real system by reusing the result of the backlog analysis for some close preceding job J_i ($i < j$). So, the backlog analysis time for real jobs may be significantly lower than that expected from the figure. Moreover, in the case where $\bar{T} = m$, the backlog analysis time slowly increases as the value of j increases, since the backlog distribution length rarely grows due to the long interarrival times for the jobs.

Figure 2(b) shows the interference analysis times (in seconds) measured for the 100th, 250th, 500th, and 1000th job, while only varying the interference degree k (i.e., the number of interfering jobs within the deadlines). The figure shows that the interference analysis time for a single job also increases in polynomial order $O(k^2m^2)$ as the interference degree increases [14]. However, the interference degree considered before the deadline is usually very small in practice. Figure 2(c) shows the interference analysis times measured for each job J_j while fixing all the other system parameters. This figure indirectly reveals the effect of the length of the p_j-backlog distribution for the j-th job to which the interference analysis is applied. As the p_j-backlog distribution length increases, the interference analysis time also increases, but slowly.

VII. Conclusions and future work

This paper has presented a framework for the stochastic analysis of periodic real-time systems which relaxes the assumption that all tasks need their worst-case execution times, assuming instead that the execution time is a random variable with a known distribution function. For the case in which all the arrival instants are deterministic, we developed a method for deriving the exact response time distribution of each task, even for systems with a maximum utilization greater than 1. Experimental results confirmed the accuracy of the proposed analysis, even for the

approximated methods. The computational complexity of the proposed analysis, polynomial with respect to the number of jobs per hyperperiod and the size of the execution time distributions, is still affordable, while allowing accurate deadline miss ratios to be derived in a much shorter time than using simulation and with greater accuracy. The stochastic analysis is directly applicable to multiprocessor systems using a partitioning scheme and common allocation algorithms such as First Fit, Best Fit, etc. In fact, a multiprocessor made up of m processors would behave like m independent uniprocessors.

When the arrival instants are not deterministic (e.g., sporadic tasks) the analysis proposed here is no longer directly applicable. This only apparently reduces the practical applicability of our framework, as in [16] we successfully investigated the possibility of obtaining "safe" *approximations* of the response time distributions instead of the exact distributions. Safe means that the probability of deadline miss derived from the approximated distribution is greater than the exact probability. It can be shown that, if each sporadic task in the system is replaced by a periodic task, with a period equal to the minimum interrelease times, our analysis can be applied to this new system, obtaining a safe approximation of the exact solution.

Future work will focus on further extensions of the framework, in order to address jitter and dependencies between the execution times.

REFERENCES

[1] L. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[2] 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*, 1989.

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

[4] 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*, 2002.

[5] 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.

[6] J. Leung and J. Whitehead, "On the Complexity of Fixed Priority Scheduling of Periodic Real-Time Tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237–250, 1982.

[7] S. Manolache, P. Eles, and Z. Peng, "Memory and Time-Efficient Schedulability Analysis of Task Sets with Stochastic Execution Times," in *Proc. of the 13th Euromicro Conference on Real-Time Systems*, Jun. 2001, pp. 19–26.

[8] A. Leulseged and N. Nissanke, "Probabilistic Analysis of Multi-processor Scheduling of Tasks with Uncertain Parameter," in *Proc. of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications*, Feb. 2003.

[9] J. P. Lehoczky, "Real-Time Queueing Theory," in *Proc. of the 17th IEEE Real-Time Systems Symposium*, 1996, pp. 186–195.

[10] 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.

[11] A. K. Atlas and A. Bestavros, "Statistical Rate Monotonic Scheduling," in *Proc. of the 19th IEEE Real-Time Systems Symposium*, 1998, pp. 123–132.

[12] J. W. S. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[13] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. LoBello, J. M. López, S. L. Min, and O. Mirabella, "Stochastic Analysis of Periodic Real-Time Systems," in *Proc. of the 23rd Real-Time Systems Symposium*, Austin, TX, USA, 2002, pp. 289–300.

[14] ——, "An exact stochastic analysis of priority-driven periodic real-time systems," Departamento de Informática, University of Oviedo, Tech. Rep., 2003, also available at http://www.atc.uniovi.es/research/AESA04.pdf.

[15] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," Dept. Computer Science, University of York, Tech. Rep. YCS 164, Dec. 1991.

[16] J. L. Díaz, J. M. López, M. García, A. M. Campos, K. Kim, and L. LoBello, "Pessimism in the stochastic analysis of real-time systems: Concept and applications," in *Proc. of the 25th Real-Time Systems Symposium*, Lisbon, Portugal, 2004, pp. 197–207.