Probabilistic Analysis of the Response Time in a Real-Time System

José Luis Díaz, José María López, Daniel Fernando García
Universidad de Oviedo
Departamento de Informática
Campus de Viesques, 33204, Gijón, Spain
{jdiaz,chechu,daniel}@atc.uniovi.es

Abstract

Classical analysis of real-time systems focuses on guaranteeing the schedulability of the system when all jobs use their worst computation time. In this report, the computation time of each job is modeled as an stochastic variable of known probability density function. Thus, we consider not only the worst-case computation time, but all the possible computation times and their probabilities. We present an algorithm to calculate the statistical distribution of the response time for each job. This result allows us to assess the feasibility of the system from a statistical point of view. We also show that the statistical analysis can be done in $O(m^3n^2)$, where m is the number of jobs and n is the maximum number of points defining the statistical distribution of their computation times. Finally, we discuss briefly how the model can be applied to periodic task sets.

INTRODUCTION

Traditional scheduling algorithms and analysis methods have focused on strict “hard” deadlines, by which a system is deemed schedulable only if every instance of every task in the system is guaranteed to meet its deadline. To achieve such a guarantee, the engineer must provide the activation period and “worst-case computation time” for each task in the system. Once this information is provided, two classic analysis methods are available [1, chap. 3]. The first is the *processor utilization analysis* [2, 3], which guarantees the schedulability whenever the total utilization is below a bound which depends on the number of tasks and the scheduling policy. The second method is the *response time analysis* [4, 5, 6, 7], which uses a different approach: the exact value of the worst-case response time is obtained for each task, which allows the analyst to assess the system feasibility by means of comparing this worst-case response time against the task deadline.

Both approaches are very restrictive, as the worst computation time assumed for each task may be too pessimistic, and the situation under which each task would suffer the maximum interference from other tasks may be very unlikely, especially if we allow for variable computation times in the tasks. Both these factors lead to very high calculated response times, which *could* occur in theory, but with little *probability* in practice. There are many soft real-time applications whose tasks have highly variable computation requirements and deadlines are not hard. For these applications, the probability of missing a deadline could serve as a measurement of its Quality of Service (QoS). Moreover, knowledge of these probabilities can still be useful for the design of hard real-time systems, as long as the application allows for a given failure rate (for example, the probability of missing a deadline could be as small as the probability of hardware failure).

Variance of computation times from their worst-case may cause an excess of capacity which can be used by soft real-time tasks. Many algorithms have been developed to schedule them together with hard real-time periodic tasks, as for example the deferred and sporadic server algorithms [8, 9] or the more recent slack-stealing algorithms [10, 11]. These algorithms focus on enhancing the responsiveness of the soft real-time tasks without jeopardizing deadlines of the hard real-time ones. However, the statistical characterization of the system's behavior is poor, as no statistical distribution is obtained for the response times of the tasks (except for the average response time of soft real-time tasks, usually by means of simulation).

A different approach to relax the assumption of fixed resource requirements is the one proposed by Atlas and Bestavros [12]. In their paper, a modification of the scheduler is introduced. This new scheduler allows a task to be executed in a variable amount of time, within a pre-fixed limit called *allowance*. Atlas and Bestavros determine the necessary allowances for guaranteeing that no more than a given percentage of deadlines is missed. Due to the need for a suitable scheduler, this analysis is not valid for the classic priority based scheduler, which is the one implemented in most real-time operating systems.

Mok and Chen [13] introduce the multiframe model, in which the computation time of a task may vary greatly from one instance to another, but this variation follows a known pattern. This behavior is modeled by specifying the computation time of a task not as a single (worst-case) number, but as a finite list of numbers from which computation times of successive instances will be generated. They investigated this model under the fixed priority preemptive scheduler, deriving new utilization bounds which improved those of Liu and Layland [2]. Whenever the system under analysis has an total utilization lower than Mok and Chen's bound, all deadlines will be met. However, when the utilization bound is exceeded, there is no clue about the probability of deadline misses for each task.

Tia *et al.*[14], address this problem by modeling the computation time of the tasks as a random variable, and extending the time-demand analysis method, substituting the sums of fixed computations by convolutions. They restrict the analysis to the first activation of the task, and assume that deadlines cannot be greater than periods. This assumption is lifted by Gardner in his PhD. thesis [15], which extends the technique of Tia *et al.*by computing the probability of deadline misses for each task instance released in the first busy-period, and picking the minimum of these probabilities. This way, a lower bound on the probability of deadline misses is found. However, this approach fails when the busy-period can have an infinite length, and indeed this will be the case when the worst-case total utilization is greater than one. Moreover, by restricting the analysis to the first busy period, the obtained probability of deadline misses is optimistic, since in the next hyperperiod the probability of having pending workload is not null, and then the response times would be greater.

As an initial step towards a more complete analysis, we formalize and extend the ideas of [14], providing mathematical proofs and detailed algorithms for finding the probability distribution functions of the response times. We present a model in which the system is not seen as a set of periodic tasks, but as a set of jobs released in a given sequence. This broader model will provides us a framework for reasoning about the stochastic behaviour of the system.

The paper is organized as follows. Section 2 introduces our model, defines some terminology, and shows a simple example. Section 3 is the core, in which the methodology of analysis is displayed, the main theorems and propositions are proved, the complexity of the algorithm is investigated and some experimental results are shown. Section 4 explores some ideas for applying our methods to the classical system model in which jobs are instances of periodic tasks. Section 5 presents the conclusions and future work.

SYSTEM MODEL

The system is modeled as a set of jobs $\{\Gamma_i\}$, each job being a three-tuple $(\lambda_i, P_i, \mathcal{C}_i)$ where λ_i is the release instant of the job, P_i is the priority under which the job runs, and $\mathcal{C}_i$ is the required computation time, which is a random variable[1] with a known probability density function (PDF), denoted by $f_{\mathcal{C}_i}$, where $f_{\mathcal{C}_i}(c) = \mathbb{P}\{\mathcal{C}_i{=}c\}$.

Note that our model does not use the classical concept of periodic tasks; each job is released only once. However, the periodic task model can be considered as a particular case, as will be shown in section 4.

Without loss of generality, we assume that release times are integers, and that job sub-indexes are ordered in increasing release times (that is $\lambda_i \leq \lambda_j$ for $i < j$). The computation time, $\mathcal{C}_i$, is a discrete random variable and its maximum value is bounded. This way, its PDF can be represented as a finite vector of values $\{f_{\mathcal{C}_i}(0), f_{\mathcal{C}_i}(1), \ldots, f_{\mathcal{C}_i}(C_i^{\max})\}$ where $C_i^{\max}$ is the worst-case computation time required by job Γ_i. No other assumption is made about the PDF of $\mathcal{C}_i$.

For example, consider the system shown in Table 1, made up of four jobs, with release times 0, 6, 9 and 17. The computation time of each job is a random discrete variable, uniformly distributed between two given values. The table shows, for instance, that the computation time of the first job, Γ_1, is $\mathrm{U}[5,7]$; this means that it can take value of 5, 6 or 7 with an equal probability of $1/3$.

[1]Throughout this paper we use a calligraphic typeface for denoting random variables, like $\mathcal{C}_i$, $\mathcal{R}_i$, etc.

Table 1: A simple example system

Job	λ_i	P_i	$f_{\mathcal{C}_i}$
Γ_1	0	10	Discrete U[5,7]
Γ_2	6	15	Discrete U[8,9]
Γ_3	9	5	Discrete U[3,5]
Γ_4	17	10	Discrete U[5,7]

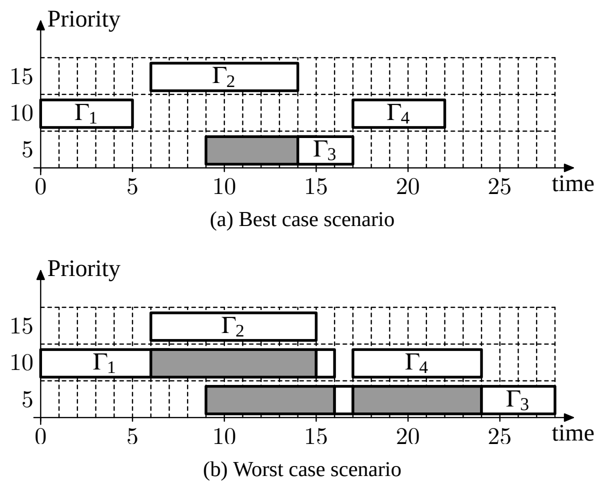


Fig. 1: Scheduling of the example in two possible scenarios

The scheduler assumed is a priority-based preemptive scheduler, i.e. it guarantees that the job which gains access to the processor (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.

We will denote by $\mathcal{R}_i$ the response time of job Γ_i, and by $\mathcal{E}_i$ the instant when this job finishes its execution, so $\mathcal{R}_i = \mathcal{E}_i - \lambda_i$. Note that $\mathcal{R}_i$ is a random variable. For example, in Fig. 1 the Gantt diagram of two possible execution scenarios are shown. In this figure, the shaded rectangles represent ready jobs unable to run because some other ready job(s) of greater priority exists, and white rectangles represent running jobs. In the first diagram, all the jobs require their minimum computation time; in this best-case scenario the job Γ_3 finishes at $\mathcal{E}_3 = 17$, with a response time of $\mathcal{R}_3 = 8$. In the second diagram, all the jobs require their maximum computation time, which causes $\mathcal{E}_3 = 28$ and thus $\mathcal{R}_3 = 19$.

Our goal is to compute the probability of occurrence of each possible response time over all the possible execution scenarios, for each job in the system, i.e.:

$$f_{\mathcal{R}_i}(r) = \mathbb{P}\{\mathcal{R}_i{=}r\} \quad i = 1, 2, \ldots \tag{1}$$

CALCULATION OF THE STATISTICAL DISTRIBUTION OF THE RESPONSE TIME

In this section we will derive a set of propositions and theorems allowing us to determine the probability density function (PDF) of the response time for any of the jobs in the system. To calculate the response time of a job, we have to take into account not only the computation time required by the job and the interference that future jobs could cause on it due to preemption, but also the pending workload not yet serviced at the instant the job is released. So, we first investigate how to determine the PDF of the pending workload at any instant.

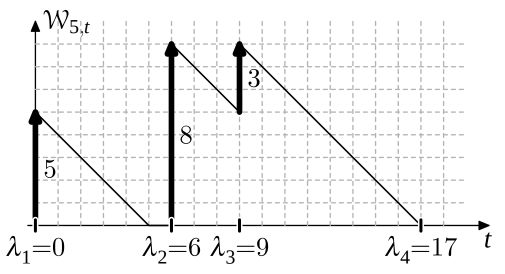
Calculation of the pending workload PDF

We will define the pending workload in relation to a certain priority level, because we are interested in its influence on the response time of a job, and there is no influence from jobs of lower priority.

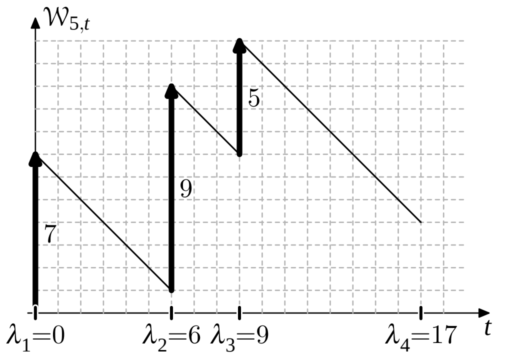
Definition 1. *The pending workload of priority level P at time t, noted as $\mathcal{W}_{P,t}$ is the sum of all computation times not yet serviced for all jobs of priority greater than or equal to P at time t.*

As an example, Fig. 2 shows the evolution in time of the pending workload of priority level 5, for the best and the worst execution scenarios of the example presented in Table 1. In this figure, each vertical arrow represents the release of a job, and the length of the arrow (labeled to its right) is the computation time required by this job.

The reader can see from this example that the pending workload of priority $P = 5$, at the instant λ_2, can take two possible values: 0 units (best case) or 1 unit (worst case). In general, $\mathcal{W}_{P,t}$ is a random variable, which can take any value at a



(a) Best case scenario



(b) Worst case scenario

Fig. 2: Pending workload of priority 5, in two different scenarios

given time. We are now interested in determining the probability of occurrence for each value.

It is easy to determine the PDF of the pending workload at any instant t' if we know the PDF of the pending workload at another previous instant t, and we know that no new jobs were released in the interim. See for example Fig. 3(a), which represents a hypothetic PDF for the pending workload of priority P, at a given instant t. Let us consider another instant, t', 6 units of time after t, and suppose that no new jobs of priority greater than or equal to P are released between t and t'. If the pending workload at t is 6 or less, the pending workload at t' will be zero. The probability for this, in the example of Fig. 3, is:

$$\mathbb{P}\{\mathcal{W}_{P,t'=0}\} = \mathbb{P}\{\mathcal{W}_{P,t\leq 6}\} = 1/27 + 3/27 + 1/27 = 5/27 \tag{2}$$

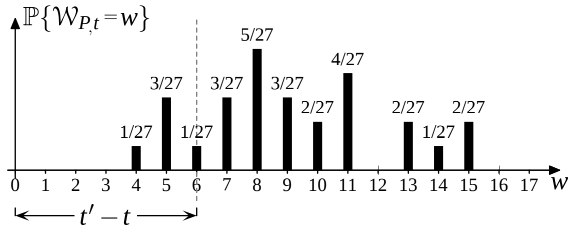
If the pending workload at t is greater than 6, the pending workload at t' will be 6 units less, as this is the time elapsed. Thus, we can build the PDF of the new pending workload by "shifting" the PDF of $\mathcal{W}_{P,t}$ 6 units to the left, and "accumulating" in the origin the values with $w \leq 0$ after the "shift". The result of this manipulation is shown in Fig. 3(b). This idea is formalized in the following proposition.

Proposition 1. *If no job with priority greater than P, is released in the interval $[t,t']$, then*

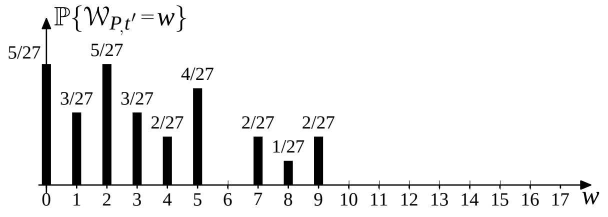
$$f_{\mathcal{W}_{P,t'}}(w) = \begin{cases} \sum_{i=0}^{t'-t} f_{\mathcal{W}_{P,t}}(i) & \textit{for } w = 0 \\ f_{\mathcal{W}_{P,t}}(w + t' - t) & \textit{for } w > 0 \end{cases} \tag{3}$$

Proof. Since no workload is added to the system between t and t', the workload in t' will be equal to the one in t, minus the elapsed time $(t' - t)$. However, the pending workload cannot be negative, so if the workload at t is less than or equal to $t' - t$, the workload at t' will be zero. From these considerations, the proof of the proposition is immediate. □

Let us now address the case in which the workload is increased by the release of a new job, at instant λ_k. The following proposition relates the PDF of the workload just after the instant λ_k with the PDF just before λ_k.



(a) For the instant t



(b) For the instant $t' = t+6$

Fig. 3: Example of the PDF of the pending workload in two different instants, t and t', separated by 6 units of time

Proposition 2. *Let $\mathcal{W}_{P,\lambda_k^-}$ denote the pending workload of priority P at instant $(\lambda_k - \varepsilon)$, when $\varepsilon \longrightarrow 0$, and let $\mathcal{W}_{P,\lambda_k^+}$ denote the workload of priority P at instant $(\lambda_k + \varepsilon)$ when $\varepsilon \longrightarrow 0$. If a single job Γ_k, with priority $P_k \geq P$ is released at instant λ_k, then:*

$$f_{\mathcal{W}_{P,\lambda_k^+}}(w) = \left(f_{\mathcal{W}_{P,\lambda_k^-}} \otimes f_{\mathcal{C}_k}\right)(w) \tag{4}$$

where $\otimes$ is the discrete convolution operator (see appendix).

Proof. The pending workload at instant λ_k suffers a discontinuity, since it is increased by the job released at λ_k, by an amount equal to its computation time $\mathcal{C}_k$. The pending workload and the computation time are both random variables, so, by elementary statistical theory, the PDF of their sum is obtained by convolution of their respective PDFs, as stated in (4). $\square$

These propositions give us an algorithm for calculating the PDF of the pending workload for a given priority level P at any instant λ_n, as follows:

1. Start at instant $t = \lambda_0$. At this instant the PDF of the pending workload is assumed to be known (in fact, the initial pending workload is usually zero).
2. Repeat until reaching the desired instant
   (a) Advance to the instant λ_k in which the next job with priority greater than or equal to P is released. Calculate the PDF of the pending workload at this instant using Proposition 1. The result is the pending workload just before the release of the job Γ_k.
   (b) Calculate the PDF of the pending workload just after the release of the new job, using Proposition 2.

Calculation of the response time PDF

In this section we will derive the method for obtaining the PDF of the response time $\mathcal{R}_{\Gamma_i}$ of an arbitrary job Γ_i released at instant λ_i (that is, the probability of $\mathcal{R}_{\Gamma_i}$ being equal to any given value).

Preliminary example

In order to clarify the problem and the method for solving it, we develop a simple example. Once the ideas are presented through this example, the method will be formalized in a theorem.

Consider the system shown in Table 2. We want to obtain the PDF of the response time for the job Γ_1, which is released at $\lambda_1 = 0$.

For this example we will assume that no pending workload is present in the system when Γ_1 is released. So, if the computation time of job Γ_1 is less than or equal to $(\lambda_2 - \lambda_1)$, its response time will be equal to this computation time, because Γ_1 will be finished before any other job in the system can preempt it. However, if the computation requirement is greater than $(\lambda_2 - \lambda_1)$, then job Γ_2 will preempt Γ_1, increasing its response time by an amount equal to $\mathcal{C}_2$. On the other hand, if $\mathcal{C}_1 > (\lambda_2 - \lambda_1)$ and $(\mathcal{C}_1 + \mathcal{C}_2) > (\lambda_3 - \lambda_1)$, then job Γ_3 will interfere, and thus $\mathcal{R}_1 = (\mathcal{C}_1 + \mathcal{C}_2 + \mathcal{C}_3)$.

The proposed approach for obtaining the PDF of $\mathcal{R}_1$ will require several steps. In step zero, we obtain the PDF of $\mathcal{R}_1$ without considering any future preemption. In step one, we modify this PDF considering the first preemption that Γ_1 could suffer. In step two, we modify it again considering the next possible preemption, and so on. Let $G^j_{\Gamma_1}$ denote the PDF obtained this way in step j.

Step zero: In our example, $G^0_{\Gamma_1}$ will be equal to $\mathcal{C}_1$, because we assumed there was no pending workload when job Γ_1 was released. In a more general case, $G^0_{\Gamma_i}$ will be the convolution of $\mathcal{C}_i$ with the pending workload of priority P_i at instant λ_i. The plot of $G^0_{\Gamma_1}$ is shown at the top of Fig. 4(a). This function, gives part of the PDF of $\mathcal{R}_1$, because:

$$\mathbb{P}\{\mathcal{R}_{\Gamma_1} = r\} = G^0_{\Gamma_1}(r) \qquad \text{for } r \le \lambda_2 - \lambda_1 \tag{5}$$

Fig. 4(a) shows that the function $G^0_{\Gamma_1}$ splits into two functions, which we will call $gl^0_{\Gamma_1}$ (*l* for *low*) and $gh^0_{\Gamma_1}$ (*h* for *high*). The splitting point is 3, which is $(\lambda_2 - \lambda_1)$. Function $gh^0_{\Gamma_1}$ can be saved as part of the desired $f_{\mathcal{R}_{\Gamma_1}}$.

Step one: Let us consider the calculation of $\mathbb{P}\{\mathcal{R}_1 = r\}$, with $r > (\lambda_2 - \lambda_1) = 3$, for example for $r = 5$. Since this event can only happen when $\mathcal{C}_1 > (\lambda_2 - \lambda_1)$, we have to calculate the probability of an intersection of events:

$$\mathbb{P}\{\mathcal{R}_{\Gamma_1} = 5\} = \mathbb{P}\{(\mathcal{C}_1 > 3) \wedge (\mathcal{C}_1 + \mathcal{C}_2 = 5)\} \tag{6}$$

This probability can be obtained by adding the probabilities of all possible cases in the intersection:

$$\mathbb{P}\{\mathcal{R}_{\Gamma_1} = 5\} = \sum_{i=4}^{5} \mathbb{P}\{\mathcal{C}_1 = i\} \cdot \mathbb{P}\{\mathcal{C}_2 = 5 - i\} \tag{7}$$

This summation is indeed the convolution, for $r = 5$, of the PDF of $\mathcal{C}_2$ with the function $gh^0_{\Gamma_1}$ shown in Fig. 4(a). This argument is true for any $r > (\lambda_2 - \lambda_1)$. On the other hand, for $r \le (\lambda_2 - \lambda_1)$ the probabilities were the ones "saved" in function $gl^0_{\Gamma_1}$ from the previous step. We can then write:

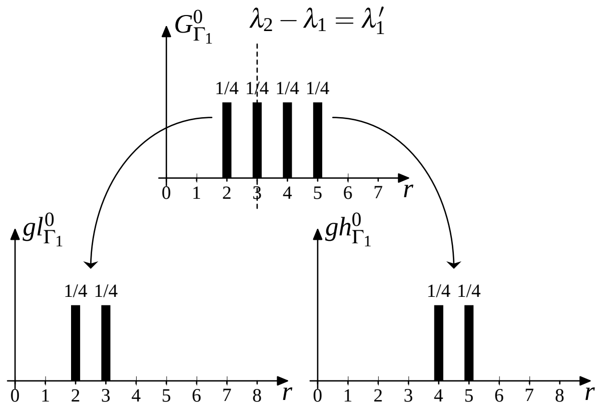
$$G^1_{\Gamma_1}(r) = gl^0_{\Gamma_1}(r) + \left(gh^0_{\Gamma_1} \otimes f_{\mathcal{C}_2}\right)(r) \tag{8}$$

Fig. 4(b) illustrates this operation.

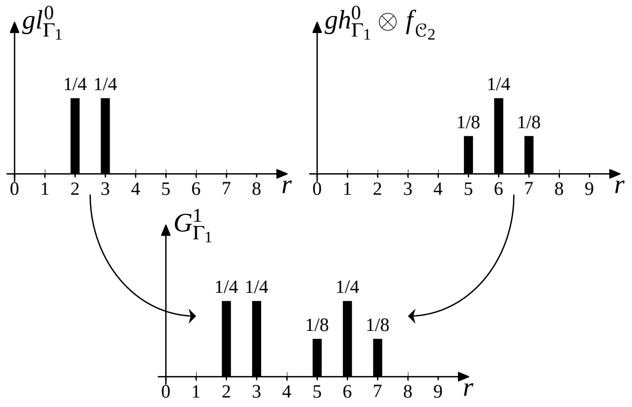
Step two: The ideas presented above are also applicable to this step. The function $G^1_{\Gamma_1}$ obtained in the previous step, splits into functions $gl^1_{\Gamma_1}$ and $gh^1_{\Gamma_1}$, the splitting point being equal to 6 (which is $\lambda_3 - \lambda_1$), as shown in Fig. 5(a). The function $gl^1_{\Gamma_1}$ gives us the first points of $G^2_{\Gamma_1}$, and the function $gh^1_{\Gamma_1}$ is convolved with the PDF of $\mathcal{C}_3$ to obtain the remaining points of $G^2_{\Gamma_1}$. This process is presented graphically in Fig. 5(b).

Table 2: Second example system

Job	λ_i	P_i	$f_{\mathcal{C}_i}$
Γ_1	0	5	Discrete U[2,5]
Γ_2	3	10	Discrete U[1,2]
Γ_3	6	15	Discrete U[1,3]



(a) “Splitting” the function $G^0_{\Gamma_1}$ at $r = 3$



(b) Construction of the function $G^1_{\Gamma_1}$

Fig. 4: Calculations for step one

Since no more jobs are released in this system, the function $G^2_{\Gamma_1}$ obtained in the last step, is the PDF of $\mathcal{R}_1$. If more jobs were released, the calculation would continue with new steps, analogous to the ones already seen.

In the following section we formalize these ideas.

Iterative method for calculating the PDF of the response time

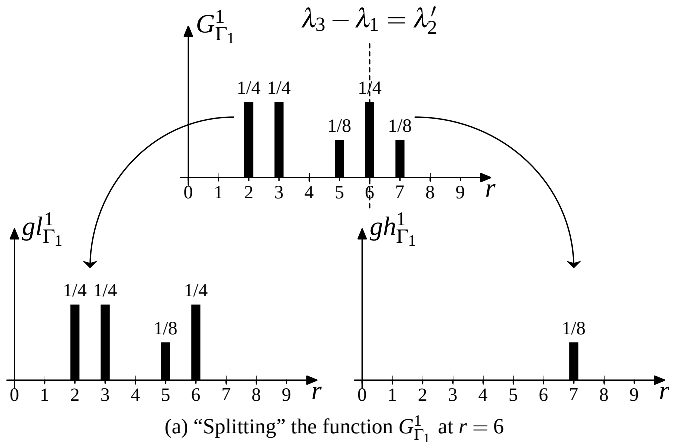
Suppose that we want to determine the PDF of the response time of an arbitrary job Γ_i, that is, the function $f_{\mathcal{R}_{\Gamma_i}}$. In order to simplify the notation, we will denote as $\Gamma'_1, \Gamma'_2, \ldots$, all jobs of priority greater than P_i which are released after the instant λ_i. Computation times of these jobs will be denoted as $\mathcal{C}'_1, \mathcal{C}'_2, \ldots$, and their release instants as $\lambda'_1, \lambda'_2, \ldots$. For completeness, we will also denote the job Γ_i as Γ'_0, its computation time as $\mathcal{C}'_0$, and its release instant as λ'_0. In the interest of simplicity, we also change the time origin to the instant λ_i. This situation is shown in Fig. 6.

We define $G^0_{\Gamma_i}$ as:

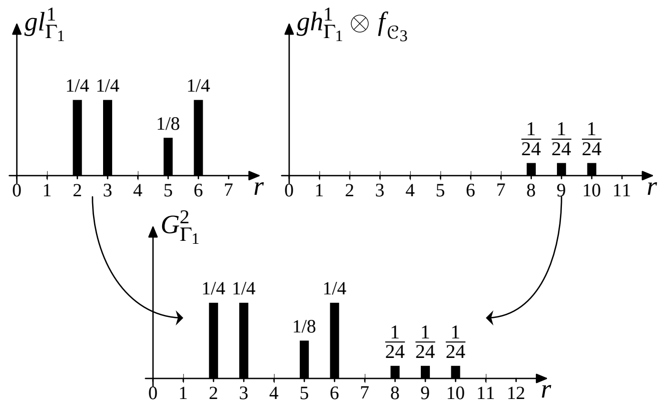
$$G^0_{\Gamma_i}(r) = \left(f_{\mathcal{W}_{P_i,\lambda_0^-}} \otimes f_{\mathcal{C}'_0}\right)(r) \tag{9}$$

This function can be understood as the PDF of $\mathcal{R}_i$ if no new jobs are released after instant $\lambda_i = \lambda'_0$. As a generalization of this concept, we define the function $G^j_{\Gamma_i}$ as the PDF of $\mathcal{R}_i$ if no new jobs are released after instant λ'_j.

Given the function $G^j_{\Gamma_i}$, we obtain from it two new functions by “splitting” $G^j_{\Gamma_i}$ at point $(\lambda'_{j+1} - \lambda'_0)$. These new functions are defined as follows (note that $(\lambda'_{j+1} - \lambda'_0) = \lambda'_{j+1}$ due to the time origin shift):



(a) "Splitting" the function $G^1_{\Gamma_1}$ at $r = 6$



(b) Construction of the function $G^2_{\Gamma_1}$

Fig. 5: Calculations for step two

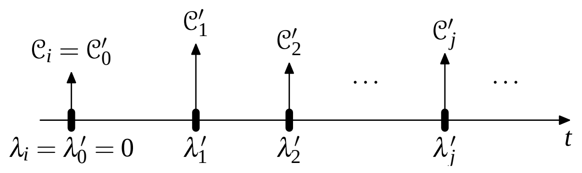


Fig. 6: Renumbering of jobs with $P_j > P_i$ and time origin shift

$$gl^j_{\Gamma_i}(r) \triangleq \begin{cases} G^j_{\Gamma_i}(r) & \text{for } r \le \lambda'_{j+1} \\ 0 & \text{for } r > \lambda'_{j+1} \end{cases} \tag{10}$$

$$gh^j_{\Gamma_i}(r) \triangleq \begin{cases} 0 & \text{for } r \le \lambda'_{j+1} \\ G^j_{\Gamma_i}(r) & \text{for } r > \lambda'_{j+1} \end{cases} \tag{11}$$

The following theorem gives a method for calculating the response time PDF of any given job.

Theorem 1. *The PDF of the response time of job Γ_i is given by function:*

$$f_{\mathcal{R}_i}(r) = G^j_{\Gamma_i}(r), \qquad \textit{for } r < \lambda'_{j+1} \tag{12}$$

being $G^j_{\Gamma_i}(r)$ calculated as

$$G^j_{\Gamma_i}(r) = gl^{j-1}_{\Gamma_i}(r) + \left(gh^{j-1}_{\Gamma_i} \otimes f_{\mathcal{C}'_j}\right)(r) \tag{13}$$

and being $G^0_{\Gamma_i}$ calculated as in (9).

Proof. Equation 12 is simply derived from the definition of $G^j_{\Gamma_i}$, since we defined this function as the PDF of the response time $\mathcal{R}_i$, if no new jobs were released after instant λ'_j. The probability of $\mathcal{R}_i$ taking a value r below λ'_{j+1} is given directly by function $G^j_{\Gamma_i}$, because for these cases the job Γ_i finishes before the instant λ'_{j+1}, and then none of the jobs arriving after λ'_j can influence it.

We will prove the recurrence relation in (13), dividing the proof in two cases, depending on the value of r.

Case 1: $r \leq \lambda'_j$

For these values, function $gh^{j-1}_{\Gamma_i}(r)$ is zero by definition, so the convolution is zero, and (13) is reduced to:

$$G^j_{\Gamma_i}(r) = gl^{j-1}_{\Gamma_i}(r) \qquad r \leq \lambda'_j \tag{14}$$

Moreover, for $r \leq \lambda'_j$ according to (10) this can be rewritten as:

$$G^j_{\Gamma_i}(r) = G^{j-1}_{\Gamma_i}(r) \qquad r \leq \lambda'_j \tag{15}$$

According to the meaning given to function $G^j_{\Gamma_i}$, the above equation implies that the probabilities of $\mathcal{R}_i$ taking any value r below λ'_j are the same, whether the job released at λ'_j is taken into account or not. This is true, because the job Γ'_j cannot preempt Γ_i, whenever $r \leq \lambda'_j$ (as this condition implies that Γ_i has finished before the release of Γ'_j).

Case 2: $r > \lambda'_j$

Let $\mathcal{R}^-$ be the response time of Γ_i without considering the influence of job Γ'_j, and $\mathcal{R}^+$ be the response time after considering its influence. The probability of $\mathcal{R}^+ = r$, when $r > \lambda'_j$ is the probability of an intersection, because two conditions must be met for such an event to occur:

1. The job Γ_i must still be active at instant λ'_j. This means that the response time *even without considering* the interference of job Γ'_j, is greater than λ'_j, that is: $\mathcal{R}^- > \lambda'_j$.
2. The sum of the response time *without considering* the interference of job Γ'_j, plus the computation requirements of this job, must be equal to r, that is: $\mathcal{R}^- + \mathcal{C}'_j = r$.

To obtain the probability of this intersection, we have to consider, from all possible cases, only those which fulfil both conditions. The sum of probabilities of these cases is the desired result. Thus, for $r > \lambda'_j$:

$$\mathbb{P}\{\mathcal{R}^+ {=} r\} = \sum_{k=\lambda'_j+1}^{\infty} \mathbb{P}\{\mathcal{R}^- {=} k\} \cdot \mathbb{P}\{\mathcal{C}'_j {=} r-k\} \tag{16}$$

By definition, $\mathbb{P}\{\mathcal{R}^- {=} k\} = G^{j-1}_{\Gamma_i}(k)$, so the above sum can be rewritten as:

$$\mathbb{P}\{\mathcal{R}^+ {=} r\} = \sum_{k=\lambda'_j+1}^{\infty} G^{j-1}_{\Gamma_i}(k) \cdot f_{\mathcal{C}'_j}(r-k) \tag{17}$$

The function $G^{j-1}_{\Gamma_i}(k)$ can be substituted by $gh^{j-1}_{\Gamma_i}(k)$, since both functions are equal in the range $[\lambda'_j+1,\infty)$, which is the range for k in the summation. Moreover, if we make this change, we can extend the lower limit of the summation to $-\infty$, because the function $gh^{j-1}_{\Gamma_i}(k)$ is zero for all $k \leq \lambda'_j$. This leads to:

$$\mathbb{P}\{\mathcal{R}^+{=}r\} = \sum_{k=-\infty}^{\infty} gh^{j-1}_{\Gamma_i}(k)\cdot f_{\mathcal{C}'_j}(r-k) = \left(gh^{j-1}_{\Gamma_i}\otimes f_{\mathcal{C}'_j}\right)(r) \qquad \text{for } r > \lambda'_j \tag{18}$$

Adding $gl^{j-1}_{\Gamma_i}(r)$ to the second member of this equation does not alter it, because $gl^{j-1}_{\Gamma_i}(r)$ is zero for all $r > \lambda'_j$. This way we obtain the same expression as in (13). On the other hand, by definition, $G^{j}_{\Gamma_i}(r)$ is the probability of the response time $\mathcal{R}_i$ taking the value r, when the interference of job Γ'_j is taken into account, that is, $\mathbb{P}\{\mathcal{R}^+{=}r\}$. Thus:

$$G^{j}_{\Gamma_i}(r) = \mathbb{P}\{\mathcal{R}^+{=}r\} = gl^{j-1}_{\Gamma_i}(r) + \left(gh^{j-1}_{\Gamma_i}\otimes f_{\mathcal{C}'_j}\right)(r) \qquad \text{for } r > \lambda'_j \tag{19}$$

$\square$

This theorem provides an iterative method for finding $\mathbb{P}\{\mathcal{R}_i{=}r\}$ for any given r. Starting from $G^0_{\Gamma_i}$, the function $G^1_{\Gamma_i}$ is calculated, and then $G^2_{\Gamma_i}$, and so on, until reaching a j such that $\lambda'_{j+1} > r$. The function $G^j_{\Gamma_i}$ obtained this way, gives the wanted probabilities for any $r < \lambda'_{j+1}$.

It is worth noting some facts of practical interest:

- Under some circumstances, it is not necessary to iterate j times to obtain $G^j_{\Gamma_i}$. If at some iteration, k, we find that $gh^k_{\Gamma_i}$ is zero in all its points, then all successive iterations will give the same function $G^k_{\Gamma_i}$. In this case, the complete PDF of $\mathcal{R}_i$ has been found, since:

$$f_{\mathcal{R}_i}(r) = G^k_{\Gamma_i}(r) \qquad \text{for all } r \tag{20}$$

- Given a deadline d_i for the job Γ_i, the probability of the response time exceeding this deadline can also be obtained from function $G^j_{\Gamma_i}$, with $d_i \leq \lambda'_{j+1}$, using the formula:

$$\mathbb{P}\{\mathcal{R}_i{>}d_i\} = 1-\mathbb{P}\{\mathcal{R}_i{\leq}d_i\} = 1-\sum_{r=0}^{d_i} G^j_{\Gamma_i}(r) \qquad \text{for } d_i \leq \lambda'_{j+1} \tag{21}$$

  This observation is of practical interest, since it means that, to obtain the probability of deadline misses, is not necessary to know the complete PDF of $\mathcal{R}_i$. It is enough to know its first d_i points, which can be obtained after j (or less) iterations of Theorem 1, j being the smallest integer such that $d_i \leq \lambda'_{j+1}$.

Computational complexity

At first glance it might seem that the calculation of the response time PDF should take all the possible interactions between all jobs into account, thus leading to a combinatorial explosion. However this is not the case; the algorithm has polynomial complexity, as will be shown in this subsection. The reason for this is that at any instant, the whole past of the system is summarized in the PDF of the pending workload for each priority level.

Let m be the number of jobs in the system, and n be the maximum number of points defining the PDF of their computation times. Let us consider the number of operations required to calculate the PDF of the response time of a job Γ_i, assuming that all other jobs in the system have a priority greater than P_i, and that their computation times are such that all of them can interfere with (preempt) the job Γ_i. This is the case which requires the maximum number of calculations, since it is necessary to perform m convolutions.

Each convolution required by the algorithm is performed between two functions: one is the PDF resulting from the previous iteration, the other is always the computation time PDF of a job, which has n points. This means that each new convolution performed will increase the size of the result in n points. In general, the result of the j-th convolution will have a size of jn points, causing the next convolution to require $O(jn^2)$ operations (see appendix). To do all the m convolutions, the number of operations required is:

$$\sum_{j=1}^{m} jn^2 = O(m^2n^2) \tag{22}$$

Since these operations have to be performed for each job in the system, and there are m jobs, the total number of operations for obtaining the response time PDF for all jobs will be $O(m^3n^2)$. Note that this is a pessimistic bound, as in practice the number of convolutions to perform for each job will be less than m. Moreover, the complexity of the convolution calculation can be reduced using the Fast Fourier Transform.

Experimental results

We have implemented a preliminary, non-optimized version of our algorithm in a simple tool. We have fed the tool with several synthetic, randomly generated, system models. Each synthetic system was composed of m jobs, whose PDF were defined by n points at maximum, m and n being the parameters for the experiment. The release instants of these jobs were selected randomly, but ensuring that the system had a high load. On average, the sum of the maximum computation time required by all the jobs was around 120% of the release instant of the last job.

The time required by the tool to solve these systems on a personal computer is shown in Fig. 7. For each point in the graph several simulations were performed, and the average time was plotted. The linearity of the logarithmic plot is to be noted, as this corroborates the polynomial complexity of the algorithm, with the exception of systems with a small number of jobs, in which the aspects of initialization and input/output of the tool prevail over the computational aspects.

APPLICATION TO THE PERIODIC TASKS MODEL

In the classic analysis, the system is modelled as a finite set of tasks $S = \{\tau_j\}$. A task is a process in the system, which periodically executes at a given priority, does its job, and finishes until the next activation. Each task τ_i can be modelled with three parameters: its period T_i, its priority P_i, and its computation time $\mathcal{C}_i$, which traditionally was a fixed amount, but will be replaced in this paper by a random variable of known PDF.

Our model can be considered a particular case of the periodic task model if we consider each instance of a task as a new job. This way, the original system S, composed of a finite number of tasks $S = \{\tau_i\}$ can be transformed in another system S' composed of an infinite number of jobs $S' = \{\Gamma_i\}$. Each task τ_i in S gives rise to an infinite number of jobs $\Gamma_{i_1}, \Gamma_{i_2}, \ldots$ all with the same priority P_i and the same computation time PDF, $f_{\mathcal{C}_i}$. The arrival instants of these jobs will be $\lambda_{i_1}, \lambda_{i_2}, \ldots$ with $(\lambda_{i_j} - \lambda_{i_{j-1}}) = T_i$. Once system S has been transformed into S', it can be analysed by the methods presented in previous sections. However, since the sequence of jobs generated this way is infinite, we have to investigate how to obtain valid and useful predictions by using only a finite subset of them.

The arrival instants of the jobs, and the PDF of their computation times follow a regular pattern, whose period is equal

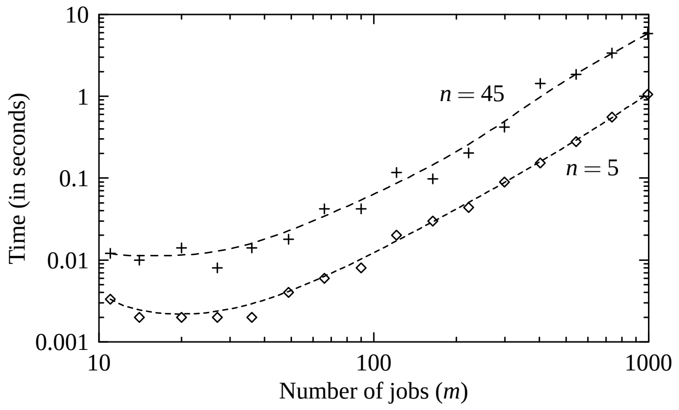


Fig. 7: Time required to run the algorithm (experimental)

to the hyperperiod of the system. This is the least common multiple (LCM) of the task periods. This regularity in the arrival of the jobs points towards a possible repetitive behaviour of the response times.

When the maximum total utilization of the system is less than one, the amount of workload generated by the jobs does not exceed the hyperperiod lenght, even in the worst-case scenario. So an hyperperiod cannot affect to the following hyperperiod, and the analysis can be restricted to a single hyperperiod.

However, if the maximum total utilization is greater than one, there exists a probability of having pending workload at the end of each hyperperiod. This modifies the initial conditions for the next hyperperiod, and thus the PDF of the response times of the jobs are different over time. We have found experimentally that, even when the maxmimum utilization factor is greater than one, if the averege utilization factor is less than one, the PDF of the pending workload at the end of each hyperperiod "converges" towards a stationary distribution. We have found a formal proof of this behaviour, and a method for obtaining the "steady state" PDF. These results are not included in this paper, for the sake of brevity.

Example with total utilization less than 1

We present a simple example with two periodic tasks. This example is based on the one presented by Lehoczky in [4] to illustrate the busy-period concept. The original example had two tasks, of periods 70 and 100, and fixed computation times of 26 and 62, respectively. The rate-monotonic policy assigns a higher priority to the first task. These settings give rise to a worst-case response time of 118 for the second task, as was calculated in [4]. Because the deadline of the task was greater than its period, its worst response time did not necessarily have to occur in its first activation. Indeed, it occurred in the fifth one.

We use the same set of tasks, with the same periods, but we now assume that the computation times are not deterministic, but random. The computation time for the first task can take two values, 25 or 26 with equal probability, and the computation time for the second task can be 61 or 62, also with equal probability. This information is summarized in Table 3.

The maximum utilization factor $U^{\max}$ is 0.991429. Since $U^{\max} < 1$, the analysis can be restricted to the first hyperperiod, which is of 700 time units. The analysis for τ_1 is trivial, since it cannot suffer interference from other tasks.

Task τ_2 will execute 7 times within the hyperperiod, and in general, the PDF of the response time will be different in each of these activations. For finding these PDFs, it is necessary to "develop" the task model into a job model. This leads to a system with 17 jobs (10 instances of τ_1 plus 7 instances of τ_2), whose release times will all be the multiples of 70 or 100. This gives the situation depicted in Fig. 8.

Note that the sub-indexes of the jobs are ordered in time, i.e. $i < j$ implies that $\lambda_i \leq \lambda_j$, as required by our model. Also note that jobs 0 and 1 arrive at same instant; so λ_0 is equal to λ_1. The computation time of jobs 0, 2, 4, 6, 7, 9, 11, 12, 14 and 16 is that of task τ_1, while the computation time of jobs 1, 3, 5, 8, 10, 13 and 15 is that of task τ_2.

Applying the techniques exposed in this paper, we calculate the PDF of the response time for jobs 1, 3, 5, 8, 10, 13 and 15, which are the activations of task τ_2, within the hyperperiod. The results are shown in Table 4 (dashes denote a

Table 3: A simple periodic system

Task	T_i	P_i	$f_{\mathcal{C}_i}$
τ_1	70	2	U[25,26]
τ_2	100	1	U[61,62]

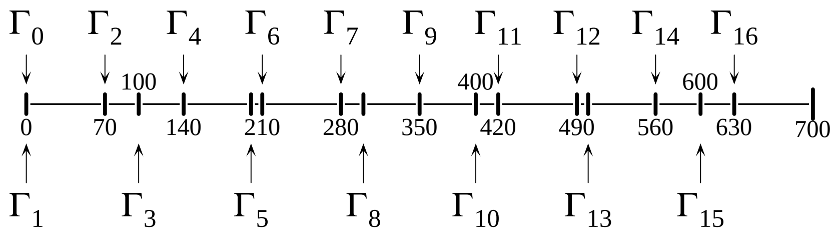


Fig. 8: Release instants for the jobs generated by the tasks in the example

Table 4: Probability of $\mathcal{R} = r$ for task τ_2 of the example

	Activation							
r	1	2	3	4	5	6	7	Average
86	—	—	—	—	0.186035	—	—	0.031006
87	—	—	—	—	0.418457	—	0.031151	0.074935
88	—	—	—	—	0.293701	—	0.155846	0.074925
89	—	—	—	—	0.078613	—	0.311974	0.065098
90	—	—	—	—	0.020019	—	0.312462	0.055414
91	—	—	—	—	—	—	0.156746	0.026124
92	—	—	—	—	—	—	0.031685	0.005281
93	—	—	—	—	—	—	0.000130	0.000022
94	—	—	—	—	—	—	0.000008	0.000001
95	—	—	—	—	—	—	—	—
96	—	—	—	—	—	—	—	—
97	—	0.031250	—	0.025391	—	—	—	0.009440
98	—	0.156250	—	0.131836	—	—	—	0.048014
99	—	0.312500	—	0.279297	—	—	—	0.098633
100	—	0.312500	—	0.307617	—	—	—	0.103353
101	—	0.156250	—	0.185547	—	0.124603	—	0.077733
102	—	0.031250	—	0.059570	—	0.374176	—	0.077499
103	—	—	—	0.009766	—	0.374939	—	0.064117
104	—	—	—	0.000977	—	0.125793	—	0.021128
105	—	—	—	—	—	0.000458	—	0.000076
106	—	—	—	—	—	0.000031	—	0.000005
107	—	—	—	—	—	—	—	—
108	—	—	—	—	—	—	—	—
109	—	—	—	—	—	—	—	—
110	—	—	—	—	—	—	—	—
111	0.125000	—	0.101562	—	—	—	—	0.037760
112	0.375000	—	0.324219	—	—	—	—	0.116537
113	0.375000	—	0.367188	—	—	—	—	0.123698
114	0.125000	—	0.171875	—	—	—	—	0.049479
115	—	—	0.031250	—	—	—	—	0.005208
116	—	—	0.003906	—	0.001465	—	—	0.000895
117	—	—	—	—	0.001587	—	—	0.000264
118	—	—	—	—	0.000122	—	—	0.000020

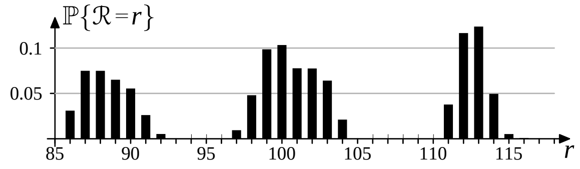


Fig. 9: Probability distribution of the response time for task τ_2, among all its possible executions

probability of zero). It is worth noting that the worst case response time (of 118 units) coincides with that calculated with the classic analysis, occurring in the fifth activation of the task, but our method provides additional information, showing us that the probability of occurrence of this worst case time is very small.

The table also shows the average probability of each response time, among all the activations of task 2. This information is shown graphically in Fig. 9. From this PDF, we can compute the probability of the response time being greater than any given value, by simple addition. For example, if the deadline of the second task were 115, the probability of missing it would be 0.001179, so we could still consider the system schedulable if we were willing to admit 0.1179% of deadline misses.

CONCLUSION AND FUTURE WORK

In this paper we have introduced stochastic response-time analysis, a model and a conceptual framework which allows statistical analysis of job response times. The model requires *a priori* knowledge of the probability density function (PDF) of the computation time required by each job. We have derived formulae for calculating the pending load PDF at any instant, and the response time PDF of any job. The statistical information given by our method can be combined with deadlines, thus obtaining the probability of deadline misses for any job in the system, which can be used as a measurement of Quality of Service. If this probability is zero for a task, then its deadline is guaranteed. In this sense our analysis subsumes the classic response-time analysis.

The analysis can be applied to the classic periodic task model, provided that the maximum total utilization is less than one. In this case, it is sufficient to perform the analysis to the jobs released within a single hyperperiod.

However, the model does not allow for uncertainty in the release instants of the jobs, and assumes all tasks to be independent. Our future work will focus on extending the analysis by taking into account the release *jitter* and the possibility of blocking jobs in the access of shared resources.

APPENDIX. DISCRETE CONVOLUTION CALCULATION

The convolution of two discrete functions f and g is defined as

$$(f \otimes g)(x) = \sum_{i=-\infty}^{\infty} f(i)g(x-i) \tag{23}$$

Note that, in all the cases presented in this paper, functions are zero for negative values of their arguments, so the lower limit of the summation can be changed to $i = 0$.

If functions f and g are defined by vectors of size n_f and n_g, the number of operations required to perform the convolution is $O(n_f n_g)$, and the resulting function will be a vector of size $(n_f + n_g - 1)$.

REFERENCES

[1] G. C. Buttazzo. *Hard Real-Time Computing Systems. Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.

[2] C. L. Liu and J. Layland. “Scheduling algorithms for multiprogramming in a hard real-time environment.” *J. ACM*, vol. 20(1), pp. 46–71, 1973.

[3] J. P. Lehoczky, L. Sha, and Y. Ding. “The rate monotonic scheduling algorithm: Exact characterization and average case behavior.” In *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 166–171. Dec 1989.

[4] J. P. Lehoczky. “Fixed priority scheduling of periodic task sets with arbitrary deadlines.” In *Proceedings 11th IEEE Real-Time Systems Symposium*, pp. 201–209. Dec 1990.

[5] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. “Hard real-time scheduling: The deadline monotonic approach.” In *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*. May 1991.

[6] K. Tindell and J. Clark. “Holistic schedulability analysis for distributed real-time systems.” *Microprocessing and Microprogramming*, vol. 50(2–3), pp. 117–134, Apr 1994.

[7] A. Burns. “Preemptive priority based scheduling: An appropiate engineering approach.” In S. H. Son, ed., *Advances in Real-Time Systems*. Prentice Hall, 1994.

[8] B. Sprunt, L. Sha, and J. P. Lehoczky. “Aperiodic task scheduling for hard real-time systems.” *Journal of Real-Time Systems*, vol. 1(1), pp. 27–60, 1989.

[9] J. P. Lehoczky, L. Sha, and J. Strosnider. “Enhanced aperiodic responsiveness in hard real-time environments.” In *Proceedings of the 8th Real-Time Systems Symposium*, pp. 261–270. Dec 1987.

[10] J. P. Lehoczky and S. Ramos-Thuel. “An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems.” In *Proceedings of the Real-Time Systems Symposium*, pp. 110–123. Dec 1992.

[11] R. I. Davis, K. Tindell, and A. Burns. “Scheduling slack time in fixed priority preemptive systems.” In *Proceedings of the Real-Time Systems Symposium*, pp. 222–231. IEEE Computer Society Press, Dec 1993.

[12] A. K. Atlas and A. Bestavros. “Statistical rate monotonic scheduling.” In *Proceedings of the 19th IEEE Real-Time Systems Symposium*. Madrid, Spain, Dec 1998.

[13] A. K. Mok and D. Chen. “A multiframe model for real-time tasks.” *IEEE Trans. Softw. Eng.*, vol. 23(10), pp. 635–645, Oct 1997.

[14] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J.-S. Liu. “Probabilistic performance guarantee for real-time tasks with varying computation times.” In *Proceedings of the Real-Time Technology and Applications Symposium*, pp. 164–173. Chicago, Illinois, May 1995.

[15] M. K. Gardner. *Probabilistic analysis and scheduling of critical soft real-time systems*. Ph.D. thesis, University of Illinois, Urbana-Champaign, 1999.