Stochastic Analysis of Buffer–less Pipelines of Real–Time Tasks

Luca Abeni
University of Trento
Via Sommarive 9, Povo
Trento (Italy)
luca.abeni@unitn.it

Daniele Fontanelli
University of Trento
Via Sommarive 9, Povo
Trento (Italy)
daniele.fontanelli@unitn.it

Luigi Palopoli
University of Trento
Via Sommarive 9, Povo
Trento (Italy)
palopoli@disi.unitn.it

Bernardo Villalba Frías
University of Trento
Via Sommarive 9, Povo
Trento (Italy)
br.villalbafrias@unitn.it

ABSTRACT

In this paper, we consider real–time applications consisting of multiple tasks, which are executed on computing cores managed by a resource reservatin scheduler. The tasks are organised in a linear topology (pipeline). The result produced by a task as a result of one of its activations is used as input for the task at the next stage of the pipeline. The time required for each execution of a task is a random variable. We assume a bufferless communication semantic, whereby a data item produced by a task is outright dropped if the consumer is not ready to execute.

Assuming a bufferless communication simplifies the computation of the probability distribution of the end–to–end delay, since when an item is correctly processed by the pipeline its accumulated delay is simply the sum of the delays incurred in each stage. However, data can be dropped at any stage if the pipeline, and this requires a precedure to compute the probability of such an event. This computation is the main problem addressed in the paper, where we also show the practical applicability of the approach through a set of experiments.

CCS Concepts

•**Computer systems organization** → **Real-time systems;** *Embedded software;*

Keywords

Stochastic Analysis, Soft Real-Time, multiprocessor

1. INTRODUCTION

Probabilistic analysis of real–time systems is increasingly used to design applications that can tolerate a limited number of missed deadlines. This kind of analysis is based on the notion of *probabilistic deadlines*, which associates a deadline with the probability that it will be respected (as opposed to "traditional" hard deadlines that must always be respected). In this setting, it is said that a real–time design guarantees a probabilistic deadline *if the deadline is met with the desired probability for a candidate choice of the scheduling parameters.*

A different viewpoint of stochastic analysis is based on the notion of "probabilistic worst–case execution time" (pWCET) [5], for which each task is associated with a stochastic worst–case execution time depending on the input data set and on random fluctuations in the system architecture (e.g., due to cache effects), and a standard hard real–time design is carried out associating a probability with the event that the guarantee is actually correctly given.

Different analysis techniques have been proposed for probabilistic guarantees assuming fixed–priority [11, 8, 9, 7] or Earliest Deadline First (EDF) schedulers [17]. When the scheduling interference among the tasks is explicitly considered, the analysis turns out to be much simpler if the scheduler permits the analysis of the behaviour of each task independently (temporal isolation). This is possible for reservation–based schedulers [1]. Different authors in the past have analysed the stochastic behaviour of this type of schedulers using numerical techniques [4, 18] or developing analytical bounds [22, 21, 20, 19].

Excluding some notable exceptions [15, 16], most of the previous research on probabilistic analysis focused on non–interacting real–time tasks (that is, applications composed of one single task). However, real–time applications implemented as pipelines of computing tasks are increasingly popular in various fields, including multimedia, computer vision and control. Examples of deterministic analysis of pipelines of real–time tasks can be found in the literature [12, 13, 14, 6], and this work extends them in two ways: first of all, probabilistic analysis (instead of deterministic analysis) is performed; second, this paper considers a *buffer–less* model for inter–task communication. According to this buffer–less communication model (that can be seen as a multi–task generalisation of the Continuous Stream model [10] recently proposed for real–time control applications), if a stage of

the pipeline generates some data when the next stage is still busy, the data is discarded (dropped).

Buffer–less communication can be interesting for real–time applications because it allows early dropping of data that will probably be delivered too late. In this way, it is possible to avoid spending computational resources in processing data that will not be delivered in time. If a real–time application can tolerate the fact that some tasks' instances are not executed (instead of being executed and completed after their deadlines), then this technique can allow to better exploit the available CPU time. Moreover, buffer–less communication simplifies the computation of the end–to–end delay of a pipeline, by making the response time of each stage of the pipeline independent from the workload of the other stages (as shown in Section 3).

After presenting and discussing this interesting model of computation (formally defining the possible behaviours of the real–time tasks implementing the various stages of the pipeline), a probabilistic analysis is presented, and the effectiveness of the proposed approach is verified through a set of experiments.

2. PROBLEM PRESENTATION AND BACKGROUND

The real–time application model considered in this paper is composed by a pipeline Γ of M tasks $\{\tau_0, ...\tau_{M-1}\}$, with τ_0 implementing the first stage of the pipeline. Each task τ_i is a stream of jobs with $J_{i,j}$ representing the j^{th} job of task τ_i (the j^{th} activation of the i^{th} stage of the pipeline).

The first task of the pipeline (τ_0) is periodically activated with period T (that is, a new job is released every T time units) to sample some input data. After processing its input, τ_0 sends the processed data to τ_1, activating it; in general, each stage of the pipeline is activated when it has some data (coming from the previous stage) to consume. The amount of CPU time needed by job $J_{i,j}$ to process its input data and activate the next stage of the pipeline is denoted as $c_{i,j}$ in this paper.

The tasks composing the pipeline are described by the activation period T of the first stage and the Probability Mass Function (PMF) of their execution times:

$$U_i(c) = \mathbf{Pr}\left\{c_{i,j} = c\right\}. \quad (1)$$

In this setting, the end–to–end response time is the interval of time between the release of the first job of the pipeline (which takes place at a multiple of the period T) and the release of the output of the same job in the last stage of the pipeline.

The problem amounts to finding, for a given deadline D, the probability that the end–to–end response time will be lower than or equal to D. We call D a probabilistic deadline.

In order to compute the probability for the pipeline to respect an end–to–end probabilistic deadline D, it is important to properly model the mechanism used to pass data in the pipeline (and to activate the various stages of the pipeline). In particular, the data produced by job $J_{i,j}$ (the j^{th} job of the i^{th} stage of the pipeline - task τ_i) can either:

- be stored in a buffer (with finite or infinite size) until task τ_{i+1} becomes ready to consume it;
- be immediately consumed by task τ_{i+1} if it is idle, or discarded if τ_{i+1} is still active when τ_i produces the data.

This paper analyses the second case in which there is no data buffering between two consecutive stages of the pipeline (buffer–less case), assuming the knowledge of the execution requirements of each task and the use of a reservation–based scheduler such as the Constant Bandwidth Server (CBS) [1]. The latter should not be seen as an academic choice, since reservation–based schedulers are now a commonplace technology distributed with main stream Linux kernels under the name of `SCHED_DEADLINE`.

We will show that the probability of having a pipeline response time smaller than the probabilistic deadline D is quite easy to compute, but it is also important to compute the probability of losing data along the pipeline, and we will show how to compute such a probability.

2.1 The CBS scheduling algorithm

The CBS algorithm works by assigning dynamic *scheduling deadlines* to the tasks and then scheduling the tasks by applying EDF on the scheduling deadlines. More details about how scheduling deadlines d_i^s associated to task τ_i are generated and updated can be found in the original papers; here, only the most important concepts are recalled:

- Each task τ_i scheduled by a CBS has two scheduling parameters Q_i^s (maximum budget) and T_i^s (server period);
- If tasks cannot migrate between CPU cores, and for each core the parameters assigned to the different tasks are such that $\sum Q_i^s/T_i^s \leq 1$, then each task is guaranteed to receive Q_i^s units of execution time within every server period T_i^s.

The latter property is of the greatest importance for probabilistic analysis since it allows us to analyse the temporal behaviour of each task in isolation, once the distribution of the inter–arrival time between two adjacent jobs activation is determined.

2.2 Model of computation

The *model of computation* is, in this context, a set of rules to decide when a task's job can start and when it can deliver its results and activate a job of the next task of the pipeline.

In "traditional" real–time systems, a task delivers its output data and activates a job of the next task of the pipeline as soon as the job processing the input data terminates. In this work, the data are not released immediately, but at a later time defined by various possible strategies (that define the model of computation):

- **Period alignment** (PA): the data are released (that is, the next job of the next stage of the pipeline is activated) when the next job arrives. For example, consider the first stage of the pipeline, which is periodically activated (with period T): if it is activated at time t_0 and it processes the received data in a time smaller than T, then the next job of the second stage is activated at time $t_0 + T$; if the first stage processes the data in a time larger than T but smaller than $2T$, than the second stage is activated at $t_0 + 2T$, etc.;
- **Greedy alignment** (GA): the next job of the $i + 1^{th}$ stage of the pipeline is activated at the next server period T_i^s. For example, if the i^{th} stage of the pipeline

is activated at time t_0 and serves the data in a time larger than $(k-1)T_i^s$ and smaller than kT_i^s, then the next stage is activated at $t_0 + kT_i^s$;

- **Greedy after period alignment** (GAPA): if a stage of the pipeline receives some data and finishes to process it before the next activation (so, no data are dropped), then the processed data are released when the next activation arrives (as in PA). If, on the other hand, some new data are dropped before the current job finishes, then the data are released at the next server period T_i^s (as in GA). For example, if the first stage (which is periodically activated with period T) is activated at time t_0 and finishes to process the data at time t_1, with $(k-1)T_0^s < t_1 - t_0 \leq kT_0^s$, then the second stage of the pipeline is activated at $t_0 + T$ if $T > kT_0^s$, otherwise it is activated at time $t_0 + kT_0^s$.

Notice that if $\forall \tau_i, T = n_0 T_i^s$ (with $n_0 \in \mathbb{N}$) then the task at the i^{th} stage of the pipeline will be activated only at times multiple of T_{i-1}^s (independently from the model of computation). This rule introduces a significant simplification in the analysis, since we can reduce the problem of stochastic analysis of a real–time task τ_i served by a CPU reservation to that of computing the probability of a job to finish in a specified *reservation period*, that is, in the time interval $(r + kT_i^s, r + (k+1)T_i^s)$, where r is the activation time of the job (see [2] for more details).

2.2.1 An example

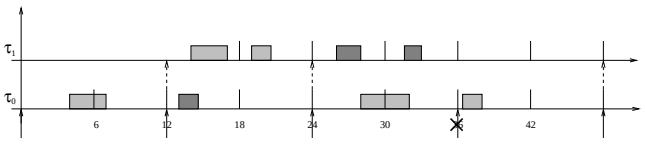

Figure 1: Example of scheduling in a pipeline composed by two tasks, with buffer–less communication, when using the PA model of computation.

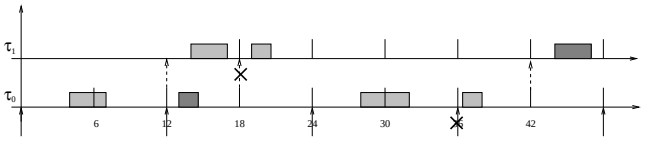

Figure 2: Same example as Figure 1, but using the GA model of computation.

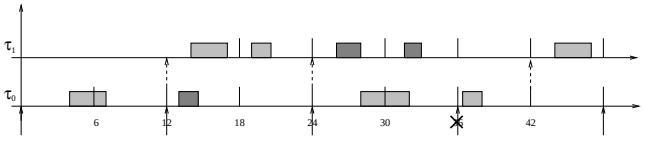

Figure 3: Same example as Figure 1, but using the GAPA model of computation.

Consider a 2–stages pipeline composed of two tasks τ_0 and τ_1. The former is activated by a flow of input data generated with a fixed period $T = 12$. The latter is activated by the data produced by τ_0. Assume that the two tasks are scheduled with two reservations with parameters $(Q_0^s = 2, T_0^s = 6)$ and $(Q_1^s = 3, T_1^s = 6)$ respectively. First of all, assume the PA model of computation, as shown in Figure 1. At time 0, the first job of the task τ_0 is released with a computation requirement of 3 time units. Since the task is idle, the job can start immediately and it will require two reservation periods to finish. The PA model of computation defers the output release to time 12. Hence, the first job of τ_1 will be activated at time 12 (when a new activation for the task τ_0 also arrives). Assume that the second activation for the task τ_0 requires 1 reservation period to finish. According to PA, the task τ_1 will be activated again at time 24 (arrival of the next job for τ_0. If task τ_1 finishes to serve its first job before time 24, the its second activation can be served without problems. If the third activation of task τ_0 (arriving at time 24) takes more than $12/6 = 2$ server periods to complete, then the next job for τ_0 will be dropped (see the dropped activation at time 36) and the next activation for task τ_1 will be at time 48.

If the GA model of computation was used instead of PA (see Figure 2, the second job of task τ_1 would have been activated at time 18, when τ_1 is still active. Hence, GA would have caused the drop of the second activation of τ_1. On the other hand, the third activation of τ_1 would have been at time 42 instead of 48, allowing τ_1 to execute first.

Finally, using GAPA (see Figure 3) the second activation of τ_1 is delivered at time 24 (avoiding to drop a job in τ_1, while the third activation is delivered at time 42.

3. ANALYSIS

Consider now the buffer–less pipeline of M tasks τ_i presented in Section 2. This section analyses the real–time behaviour of the pipeline when the first stage uses the PA, GA, or GAPA model of computation, while the other stage of the pipeline use the GA model (the analysis for the other two models of computation is similar and is not described for the sake of simplicity).

As explained in Section 2.2.1, if τ_{i+1} is busy when τ_i produces some data, the data is lost (and *the activation is dropped*). As a result of the absence of buffers, computing the end–to–end delay for a periodic activation is quite easy, since the amount of time to be served when a new job is activated is always equal to the job's execution time. To prove this important property, it is useful to provide some more formal definitions. In particular, as shown in [3, 4] the amount of time $v_{i,j}$ to be served when a job $J_{i,j}$ is activated (also named *backlog*) can be computed as:

$$v_{i,j} = \max\{0, v_{i,j-1} - z_{i,j} Q_i^s\} + c_{i,j}, \qquad (2)$$

where $r_{i,j}$ is the arrival time of job $J_{i,j}$ and $z_{i,j}$ is the number of server periods T_i^s between the arrival of $J_{i,j-1}$ and $J_{i,j}$ (more formally, $z_{i,j} = \left\lceil \frac{r_{i,j} - r_{i,j-1}}{T_i^s} \right\rceil$).

Based on this definition it is possible to prove the following lemma:

LEMMA 1. *When considering a pipeline of tasks served by resource reservations and without buffers between the stages of the pipeline, the backlog of each task when a new job arrives is equal to the job's execution time:* $v_{i,j} = c_{i,j}$

PROOF. This can be easily proved by contradiction, assuming that $J_{i,j}$ is not dropped and showing that $v_{i,j} \neq c_{i,j}$ contraddicts this hypothesis.

If $J_{i,j}$ is not dropped (hence, it starts its execution), according to Equation 2 $v_{i,j} \neq c_{i,j}$ implies that

$$v_{i,j-1} - z_{i,j}Q_i^s > 0 \Rightarrow v_{i,j-1} > z_{i,j}Q_i^s.$$

But since $z_{i,j} = \left\lceil \frac{r_{i,j}-r_{i,j-1}}{T_i^s} \right\rceil$, we have

$$\begin{aligned} & v_{i,j-1} - \left\lceil \frac{r_{i,j}-r_{i,j-1}}{T_i^s} \right\rceil Q_i^s > 0 \Rightarrow \\ & \Rightarrow \left\lceil \frac{r_{i,j}-r_{i,j-1}}{T_i^s} \right\rceil Q_i^s < v_{i,j-1} \Rightarrow \\ & \Rightarrow \frac{r_{i,j}-r_{i,j-1}}{T_i^s} < \frac{v_{i,j-1}}{Q_i^s} \Rightarrow \\ & \Rightarrow r_{i,j} < r_{i,j-1} + \frac{v_{i,j-1}}{Q_i^s}T_i^s < r_{i,j-1} + \left\lceil \frac{v_{i,j-1}}{Q_i^s} \right\rceil T_i^s \end{aligned}$$

Now, notice that $\left\lceil \frac{v_{i,j-1}}{Q_i^s} \right\rceil T_i^s$ is the response time for $J_{i,j-1}$, hence $r_{i,j} < r_{i,j-1} + \left\lceil \frac{v_{i,j-1}}{Q_i^s} \right\rceil T_i^s = f_{i,j-1}$ (i.e., the finishing time of job $j-1$), contradicting the hypothesis that the activation of job $J_{i,j}$ was not dropped. □

Since $v_{i,j} = c_{i,j}$, the response time for a non–dropped job $J_{i,j}$ is $\delta_{i,j} = \left\lceil \frac{v_{i,j}}{Q_i^s} \right\rceil T_i^s = \left\lceil \frac{c_{i,j}}{Q_i^s} \right\rceil T_i^s$, hence $f_{i,j} = r_{i,j} + \left\lceil \frac{c_{i,j}}{Q_i^s} \right\rceil T_i^s$ (notice that this requires that τ_i is not busy at time $r_{i,j}$). If the GA model of computation is used, since there is no buffering $r_{i,j} = f_{i-1,j}$, hence $f_{i,j} = f_{i-1,j} + \left\lceil \frac{c_{i,j}}{Q_i^s} \right\rceil T_i^s$.

It is now possible to define $\widetilde{c}_{i,j} = \left\lceil \frac{c_{i,j}}{Q_i^s} \right\rceil T_i^s$, writing the finishing time of a job as

$$f_{i,j} = f_{i-1,j} + \widetilde{c}_{i,j}.$$

Hence, the end–to–end delay can be described as

$$\delta_{i,j} = \sum_{i=0}^{M-1} \widetilde{c}_{i,j}. \tag{3}$$

Since the execution time of a job does not depend on the job index ($\mathbf{Pr}\{c_{i,j} = c\}$ does not depend on j), and since the execution times of stage i are independent from the execution times of stage $k \neq i$, $\mathbf{Pr}\{\delta_{i,j} = d\}$ is given by the convolution of the PMFs $U'_z(c) = \mathbf{Pr}\{\widetilde{c}_{z,j} = cT_i^s\}$ of $\widetilde{c}_{z,j}$ for all the pipeline stages from 0 to i.

This result can be adapted to support the case in which the first stage of the pipeline uses a different model (PA or GAPA); the only requirement is that the GA model of computation is used in all the stages of the pipeline $i \neq 0$. The only modification needed is a different definition of $\widetilde{c}_{0,j}$ (instead of using $\widetilde{c}_{0,j} = \left\lceil \frac{c_{0,j}}{Q_0^s} \right\rceil T_0^s$): if stage 0 uses the PA model it is possible to define $\widetilde{c}_{0,j} = \left\lceil \frac{c_{0,j}}{(T/T_0^s)Q_0^s} \right\rceil T$; if stage 0 uses GAPA, it is possible to define $\widetilde{c}_{0,j} = T$ if $c_{0,j} \leq (T/T_0^s)Q_0^s$ and $\widetilde{c}_{0,j} = \left\lceil \frac{c_{0,j}}{Q_0^s} \right\rceil T_0^s$ if $c_{0,j} > (T/T_0^s)Q_0^s$.

Of course, there is a cost to be paid to achieve such a simple computation of the response time (and such an independence between the response times of the various stages of the pipeline): since data can be dropped (if τ_i is busy when τ_{i-1} generates the data), it is not guaranteed that the j^{th} periodic activation of task τ_0 (the first stage of the pipeline) generates some output on τ_{M-1} (the end of the pipeline).

Hence, the real–time performance of the application cannot be described only by the distribution of the end–to–end response times for activations that are not dropped, but the probability to drop some periodic activation (in any of the stages of the pipeline) must be computed too, as shown in the following of this section.

3.1 Drop Probability for the First Stage of the Pipeline

For the sake of simplicity, in what follows we will assume that the same reservation period is used for all the stages of the pipeline: $\forall \tau_i, T_i^s = T^s$.

In order to compute the probability to drop a job, it can be useful to define the event $E_{i,j}$ as "the j^{th} job has been executed in the i^{th} stage". Then, the probability of dropping the j^{th} job at the i^{th} stage can be computed as $M_{i,j} = 1 - \mathbf{Pr}\{E_{i,j}\}$. In the following it will also be useful to define $m_i = \left\lceil \frac{\max_j(\widetilde{c}_{i,j})}{T} \right\rceil$.

Since in the first stage of the pipeline the jobs are periodically activated with period $T = n_0 T^s$, we have

$$\begin{aligned} M_{0,j} &= \sum_{k=1}^{m_0-1} \mathbf{Pr}\{E_{0,j-k} \wedge \widetilde{c}_{0,j-k} > kT\} \\ &= \sum_{k=1}^{m_0-1} (1 - M_{0,j-k})\mathbf{Pr}\{\widetilde{c}_{0,j-k} > kT\}. \end{aligned} \tag{4}$$

If the process $\widetilde{c}_{0,j}$ is stationary, it is easy to show that the probability to drop a job does not depend on the job index. Hence Equation 4 can be rewritten as

$$M_{0,j} = \sum_{k=1}^{m_0-1} (1 - M_{0,j})\mathbf{Pr}\{\widetilde{c}_{0,j-k} > kT\},$$

which leads to

$$M_{0,j} = \frac{\sum_{k=1}^{m_0-1} \mathbf{Pr}\{\widetilde{c}_{0,j-k} > kT\}}{1 + \sum_{k=1}^{m_0-1} \mathbf{Pr}\{\widetilde{c}_{0,j-k} > kT\}}. \tag{5}$$

3.2 Drop probabiity for the Generic Stage of the Pipeline

Computing the probability to drop a job for the next stages of the pipeline is more complex, because jobs are not periodically activated anymore. Hence, this probability depends on the probability distribution of the jobs inter–arrival times $T_{i,j}$. More formally, $T_{i,j}$ is defined as the time between the arrivals of jobs $J_{i,j}$ and $J_{i,j+1}$: $T_{i,j} = r_{i,j+1} - r_{i,j}$ (difference between the activation time of the j^{th} and the activation time of the $(j+1)^{th}$ job at stage i). Notice that using the PA, GA or GAPA models of computation the inter-arrival times will be multiple of the reservation period T^s.

We will show later in the section how the probability $\mathbf{Pr}\{T_{i,j} = hT^s\}$ can be computed. Assuming it known, at the moment, it is possible to compute the probability to drop a job as stage i as follows. To compute the probability of dropping the j^{th} job at the i^{th} stage of the pipeline, it is convenient to start by first assuming that the job has been correctly executed. To this end, let us define $n_i = \left\lceil \frac{\max_j(\widetilde{c}_{i,j})}{T^s} \right\rceil$.

Similarly to (4), we have

$$M_{1,j} = \sum_{k=1}^{n_1-1} \mathbf{Pr}\left\{E_{1,j-k} \wedge \widetilde{c}_{1,j-k} > \sum_{l=1}^{k} T_{1,j-l}\right\} = \sum_{k=1}^{m_1-1} (1 - M_{1,j-k})\mathbf{Pr}\left\{\widetilde{c}_{1,j-k} > \sum_{l=1}^{k} T_{1,j-l}\right\}, \quad (6)$$

that again assuming the process stationary,

$$M_{1,j} = \sum_{k=1}^{n_1-1} (1 - M_{1,j})\mathbf{Pr}\left\{\widetilde{c}_{1,j-k} > \sum_{l=1}^{k} T_{1,j-l}\right\},$$

that finally yields to

$$M_{1,j} = \frac{\sum_{k=1}^{n_1-1} \mathbf{Pr}\left\{\widetilde{c}_{1,j-k} > \sum_{l=1}^{k} T_{1,j-l}\right\}}{1 + \sum_{k=1}^{n_1-1} \mathbf{Pr}\left\{\widetilde{c}_{1,j-k} > \sum_{l=1}^{k} T_{1,j-l}\right\}}. \quad (7)$$

The Equation (7) is more involved than (5) presented for the first stage because the inter–arrival times are time–varying, while the period T is fixed. In order to compute all the summands in (7), we first start with the case of $k = 1$, that can be rewritten as

$$\mathbf{Pr}\left\{\widetilde{c}_{1,j-1} > T_{1,j-1}\right\} = \sum_{q_1=1}^{n_1-1} \mathbf{Pr}\left\{\widetilde{c}_{1,j-1} > q_1 T^s\right\} \cdot \cdot \mathbf{Pr}\left\{T_{1,j-1} = q_1 T^s\right\},$$

that represents all the possible situations in which the computation time is greater than the available inter–arrival time $T_{1,j-1}$. For $k = 2$ we have,

$$\mathbf{Pr}\left\{\widetilde{c}_{1,j-2} > T_{1,j-1} + T_{1,j-2}\right\} = \sum_{q_1=1}^{n_1-2} \mathbf{Pr}\left\{T_{1,j-2} = q_1 T^s\right\} \cdot \cdot \sum_{q_2=q_1+1}^{n_1-1} \mathbf{Pr}\left\{T_{1,j-1} = (q_2 - q_1)T^s\right\} \mathbf{Pr}\left\{\widetilde{c}_{1,j-2} > q_2 T^s\right\},$$

where all the possible combinations of two inter–arrival times for two consecutive jobs $T_{1,j-1}$ and $T_{1,j-2}$ are considered. It then follows that, for the generic case comprising k inter–arrival times

$$\mathbf{Pr}\left\{\widetilde{c}_{1,j-k} > \sum_{l=1}^{k} T_{1,j-l}\right\} = \sum_{q_1=1}^{n_1-k} \mathbf{Pr}\left\{T_{1,j-k} = q_1 T^s\right\} \cdot \cdot \sum_{q_2=q_1+1}^{n_1-(k-1)} \mathbf{Pr}\left\{T_{1,j-k+1} = (q_2 - q_1)T^s\right\} \cdot \cdot \sum_{q_3=q_2+1}^{n_1-(k-2)} \mathbf{Pr}\left\{T_{1,j-k+2} = (q_3 - q_2)T^s\right\} \cdots \cdots \sum_{q_k=q_{k-1}+1}^{n_1-1} \mathbf{Pr}\left\{T_{1,j-1} = (q_k - q_{k-1})T^s\right\} \mathbf{Pr}\left\{\widetilde{c}_{1,j-k} > q_k T^s\right\}, \quad (8)$$

It is worthwhile to note that the combination of the summations of the k inter–arrival periods can be radically simplified making use of the convolutions. Indeed, by defining $\mathbf{Conv}(i, q, k, j-k)$ as the q–th component of the discrete convolution of k inter–arrival PMFs starting from $j-k$ at the i–th stage, we have that

$$\mathbf{Pr}\left\{\sum_{l=1}^{k} T_{1,j-l} = qT^s\right\} = \mathbf{Conv}(1, q, k-1, j-k).$$

As a consequence, (8) boils down to

$$\mathbf{Pr}\left\{\widetilde{c}_{1,j-k} > \sum_{l=1}^{k} T_{1,j-l}\right\} = \sum_{q=1}^{n_1-1} \mathbf{Conv}(1, q, k-1, j-k)\, \mathbf{Pr}\left\{\widetilde{c}_{1,j-k} > qT^s\right\}. \quad (9)$$

At this point, the only missing piece is the computation of the job inter–arrival times $\mathbf{Pr}\left\{T_{1,j-1} = hT^s\right\}$ for the various stages of the pipeline.

3.2.1 Job Inter–Arrival Times: Second Stage of the Pipeline

Before computing the probability distribution of job inter–arrival times at a generic stage i, let us first consider the second stage ($i = 1$). Depending on the computation model adopted at the first stage, the second stage behaviour changes accordingly. In this paper we analyse the three different models presented in Section 2: PA, GA and GAPA.

According to the chosen policy, we have:

- **PA**: For $h = wn_0$, $w = 1, \ldots, m_0$, we have:

  $$\mathbf{Pr}\left\{T_{1,j} = hT^s\right\} = \mathbf{Pr}\left\{(w-1)n_0T^s < \widetilde{c}_{0,j+1} \leq hT^s\right\},$$

  whereas for $h \neq wn_0$ the probability is zero;

- **GA**: For $h = 1, \ldots, n_0m_0$:

  $$\mathbf{Pr}\left\{T_{1,j} = hT^s\right\} = \sum_{k=\max(1,h-n_0+1)}^{h} \mathbf{Pr}\left\{\widetilde{c}_{0,j+1} = kT^s\right\} \cdot \cdot \sum_{l=1}^{m_0} \mathbf{Pr}\left\{\widetilde{c}_{0,j} = (l-(h-k))n_0T^s\right\};$$

- **GAPA**: To compute the inter–arrival times distribution for this model of computation, it is necessary to consider three different cases:

  1. If $h < n_0$, $\mathbf{Pr}\left\{T_{1,j} = hT^s\right\} = 0$. For $h = n_0$, it is:

     $$\mathbf{Pr}\left\{T_{1,j} = hT^s\right\} = \mathbf{Pr}\left\{\widetilde{c}_{0,j+1} \leq n_0T^s\right\}\Gamma,$$

     where

     $$\Gamma = \mathbf{Pr}\left\{\widetilde{c}_{0,j} \leq n_0T^s\right\} + \sum_{l=2}^{m_0} \mathbf{Pr}\left\{\widetilde{c}_{0,j} = ln_0T^s\right\}.$$

  2. If $n_0 < h < 2n_0$, one gets

     $$\mathbf{Pr}\left\{T_{1,j} = hT^s\right\} = \mathbf{Pr}\left\{\widetilde{c}_{0,j+1} = hT^s\right\}\Gamma + \mathbf{Pr}\left\{\widetilde{c}_{0,j+1} \leq n_0T^s\right\} \cdot \cdot \left(\sum_{l=2}^{m_0} \mathbf{Pr}\left\{\widetilde{c}_{0,j} = [(l+1)n_0 - h]T^s\right\}\right) + \sum_{k=1}^{h-n_0-1} \mathbf{Pr}\left\{\widetilde{c}_{0,j+1} = (h-k)T^s\right\} \cdot \cdot \sum_{l=2}^{m_0} \mathbf{Pr}\left\{\widetilde{c}_{0,j} = [(l+1)n_0 - (h-k)]T^s\right\}.$$

3. Finally, for $h \geq 2n_0$, we have

$$\mathbf{Pr}\left\{T_{1,j} = hT^s\right\} = \mathbf{Pr}\left\{\widetilde{c}_{0,j+1} = hT^s\right\}\Gamma + \sum_{k=1}^{\tilde{h}} \mathbf{Pr}\left\{\widetilde{c}_{0,j+1} = (h-k)T^s\right\} \cdot \cdot \left(\sum_{l=2}^{m_0} \mathbf{Pr}\left\{\widetilde{c}_{0,j} = (ln_0 - k)T^s\right\}\right),$$

where

$$\tilde{h} = \min\left[h - \left(\left\lfloor \frac{h}{n_0} \right\rfloor - 1\right) n_0 - 1, n_0 - 1\right].$$

3.2.2 Job Inter–Arrival Times: the Generic Stage of the Pipeline

For the generic i^{th} stage, the inter–arrival times depend on the model of computation used by the $i-1^{th}$ stage: PA, GA or GAPA; for the sake of simplicity, this paper presents the analysis only for the GA model, since the analysis for the other two models of computation is similar. As it is clear from the previous analysis, the key quantity to be computed starting from the second stage on is the probability of having a certain job inter–arrival time. Indeed, once the probabilities of the $T_{i,j}$ are available, we can directly apply (6), (7) and (9) to derive the probability of dropping a job. Before going into the details, we define I_i as the maximum number of server periods T^s for the inter–arrival times for the i–th stage.

Considering the GA model of computation, the probability of $T_{i,j} = hT^s$ can be computed by looking at two consecutive jobs executing on stage $i-1$ and considering all the possible combinations of inter–arrivals and jobs execution times so that the distance between the end of the two jobs is hT^s:

$$\begin{aligned}
&\mathbf{Pr}\left\{T_{i,j} = hT^s\right\} = \\
&\sum_{q_1=1}^{I_{i-1}} \mathbf{Conv}\left(i-1, q_1, 1, j\right) \sum_{l=1}^{q_1} \mathbf{Pr}\left\{\widetilde{c}_{i-1,j} = lT^s\right\} \cdot \\
&\cdot \mathbf{Pr}\left\{\widetilde{c}_{i-1,j+1} = (h-q_1+l)T^s\right\} + \\
&\sum_{q_1=1}^{I_{i-1}} \mathbf{Conv}\left(i-1, q_1, 1, j\right) \sum_{q_2=q_1+1}^{I_{i-1}+q_1} \mathbf{Pr}\left\{T_{i-1,j+1} = (q_2-q_1)T^s\right\} \cdot \\
&\cdot \sum_{l=q_1+1}^{q_2} \mathbf{Pr}\left\{\widetilde{c}_{i-1,j} = lT^s\right\} \mathbf{Pr}\left\{\widetilde{c}_{i-1,j+2} = (h-q_2+l)T^s\right\} + \\
&\sum_{q_1=2}^{2I_{i-1}} \mathbf{Conv}\left(i-1, q_1, 2, j\right) \sum_{q_2=q_1+1}^{I_{i-1}+q_1} \mathbf{Pr}\left\{T_{i-1,j+2} = (q_2-q_1)T^s\right\} \cdot \\
&\cdot \sum_{l=q_1+1}^{q_2} \mathbf{Pr}\left\{\widetilde{c}_{i-1,j} = lT^s\right\} \mathbf{Pr}\left\{\widetilde{c}_{i-1,j+3} = (h-q_2+l)T^s\right\} + \cdots + \\
&\sum_{q_1=k}^{kI_{i-1}} \mathbf{Conv}\left(i-1, q_1, k, j\right) \sum_{q_2=q_1+1}^{I_{i-1}+q_1} \mathbf{Pr}\left\{T_{i-1,j+k} = (q_2-q_1)T^s\right\} \cdot \\
&\cdot \sum_{l=q_1+1}^{q_2} \mathbf{Pr}\left\{\widetilde{c}_{i-1,j} = lT^s\right\} \mathbf{Pr}\left\{\widetilde{c}_{i-1,j+k+1} = (h-q_2+l)T^s\right\} + \cdots
\end{aligned}$$

where the summation is performed at most I_i times. Since I_i is not known upfront, an upper bound is given by $I_i = I_{i-1} - \left\lceil \frac{\min_j(\widetilde{c}_{i-1,j})}{T^s} \right\rceil + n_{i-1}$. It is worthwhile to note that this summation very much recalls (9).

Finally, notice that the probability of dropping the job j at any of the stage up to the stage k is given by the sum of the conditional probabilities, i.e.,

$$\begin{aligned}
M_{\leq k,j} &= M_{0,j} + (1-M_{0,j})M_{1,j} + (1-M_{0,j})(1-M_{1,j})M_{2,j} + \\
&+ \cdots + (1-M_{0,j})\cdots(1-M_{k-1,j})M_{k,j} = \\
&= \sum_{i=1}^{k} M_{i,j} + \sum_{i=0}^{k}(-1)^i \prod_{q=0}^{i} M_{q,j}.
\end{aligned}$$

4. EXPERIMENTS

In order to show the practical applicability of the presented approach, we considered a pipeline of four tasks derived by a computer vision application. The application is used on a mobile robot to identify the boundaries of a lane and estimate the position of the robot by using a web–cam mounted on the chassis of the robot.

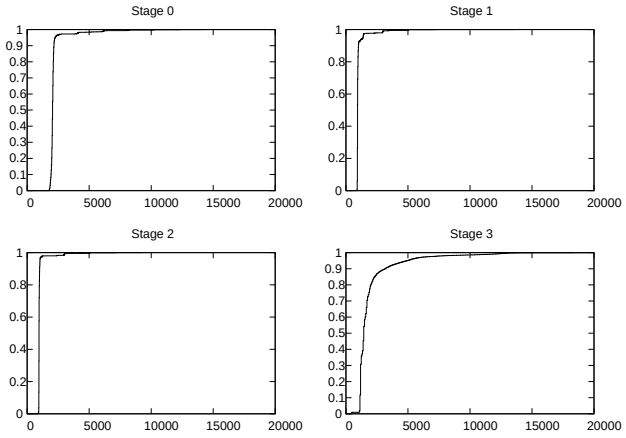

Figure 4: Cumulative Distribution Functions (CDFs) of the execution times for the 4 stages of the pipeline.

The first stage of the pipeline is activated with period $T = 8$ ms, and the four stages have execution times distributed as in Figure 4. The WCETs are 20.385 ms (for stage 0), 13.557 ms (for stage 1), 9.310 ms (for stage 2), and 51.695 ms (for stage 3); the minimum execution times are 1.713 ms (for stage 0), 0.870 ms (for stage 1), 0.840 ms (for stage 2), and 0.463 ms (for stage 3); the average execution times are 2.158 ms (for stage 0), 1.023 ms (for stage 1), 1.018 ms (for stage 2), and 1.954 ms (for stage 3). The execution times distributions have been measured by instrumenting a real implementation of the application running on a mobile robot controlled by a WandBoard Quad[1].

The tasks implementing the four stages of the pipeline are scheduled using the `SCHED_DEADLINE` policy on a 4.1.5 Linux kernel. Since using a too small server period is not feasible in practice, the server period has been set to $T^s = 8$ ms (equal to the stage 0 period) for all of the four tasks.

As a first experiment, the pipeline has been analysed assigning $Q_0^s = 2$ ms, $Q_1^s = 1$ ms, $Q_2^s = 1$ ms, and $Q_3^s = 1.5$ ms. Notice that the system is heavily overloaded, since the maximum budget of each task has been set to a value smaller than the average execution time of the task. If the pipeline

[1]http://www.wandboard.org

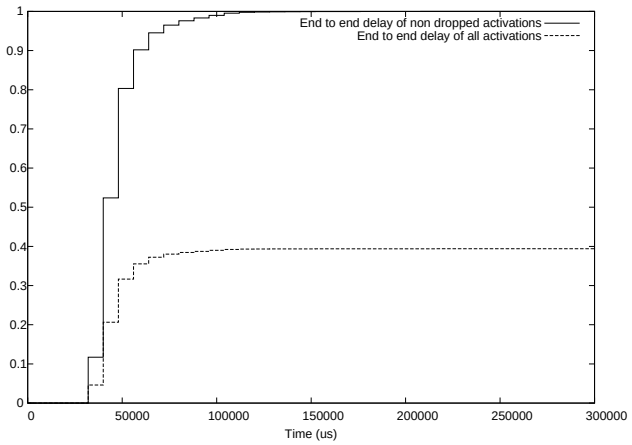

Figure 5: CDF of the end–to–end delay for a pipeline with 4 stages. The execution times are distributed as in Figure 4; the server period is equal to the period of stage 0 and is $T^s = 8$ ms; the maximum budgets are $Q_0^s = 2$ ms, $Q_1^s = Q_2^s = 1$ ms $Q_3^s = 1.5$ ms.

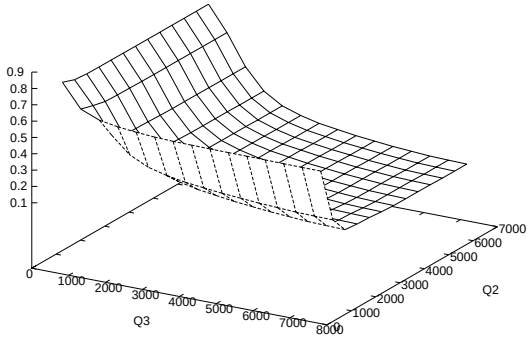

Figure 6: Probability to drop an activation for a pipeline with 4 stages, as a function of Q_2^s and Q_3^s (after fixing $Q_0^s = 2.5$ ms and $Q_1^s = 2$ ms). The execution times are distributed as in Figure 4.

was implemented using queues between the various stages, the end–to–end delay of the pipeline would have been unbounded; however, using a buffer–less communication model allows to have a bounded end–to–end delay (at the cost of dropping some activations). Figure 5 shows the CDF of the end–to–end delay experienced when using the GA model of computation. The probability to drop an activation has been computed as 0.606, hence the figure also plots the CDF of the end–to–end delay for all the activations of stage 0. This second curve does not arrive to 1, but it is scaled up to $1 - 0.606 = 0.394$, because the probability for an activation not to be dropped is only 0.394. Notice that these results have been analytically computed as shown in Section 3, but the system has also been simulated to verify that the simulative results match the analytical ones.

By looking at Figure 5 it is possible to appreciate the fact that even if the system is overloaded the data fed in the pipeline have almost 40% probability of being processed in less than 100 ms. As previously noticed, this result cannot be obtained if the pipeline is not allowed to drop some data/activations.

Of course, to achieve better performance the reservations have to be dimensioned in a more reasonable way (setting Q_i^s to a value larger than the average computation time of stage i). To show this, Figure 6 plots the probability to drop an activation as a function of Q_2^s and Q_3^s, after setting $Q_1^s = 2.5$ ms and $Q_2^s = 2$ ms. By looking at the figure it is possible to notice that increasing Q_2^s from 0.5 ms to 1.5 ms greatly decreases the probability to drop an activation, but increasing it to more than 2.5 ms only slightly improves the performance. This effect is less visible on Q_3^s, but it can be noticed that for values larger than 5 ms the variations in the probability to drop an activation are small enough. As an example, the probability to drop an activation with $Q_2^s = 1.5$ ms and $Q_3^s = 5$ ms is 0.163; increasing Q_2^s to 4.5 ms the probability decreases to 0.140 (further increasing Q_2^s only decreases this value for less than 0.0001), and increasing Q^3 to 6.5 ms the probability decreases to 0.113.

Focusing on one point of the curve, Figure 7 plots the CDF of the end–to–end response times for $Q_2^s = 4.5$ ms and $Q_3^s = 6.5$ ms (as said, the probability to drop an activation, with these parameters, is 0.113).

5. CONCLUSIONS

In this paper we have presented a stochastic analysis for buffer–less pipelines of tasks scheduled through a reservation–based scheduler. Different models of computation (PA, GA, GAPA) have been introduced, and the GA model has been formally analysed. The correspondence between analytical results and the model have been verified through simulation, and some examples have been presented to show the effectiveness and usefulness of the presented analysis.

In the future work, we plan to extend the analysis to different types of topologies and communication semantics, and to compare the real–time performance of the buffer–less model presented in this paper with more traditional communication models.

6. ACKNOWLEDGEMENTS

The research leading to the results presented in the paper has been partially supported by the European Commission H2020 programme, through the ACANTO Research and Innovation Action, Grant number 643644.

7. REFERENCES

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[2] L. Abeni and G. Buttazzo. Qos guarantee using probabilistic dealines. In *Proceedings of the IEEE Euromicro Conference on Real-Time Systems*, York, England, June 1999.

[3] L. Abeni and G. Buttazzo. Stochastic analysis of a reservation-based system. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium.*, San Francisco, California, April 2001.

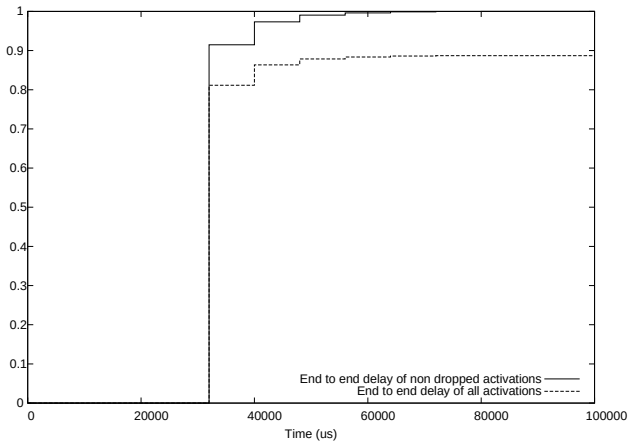

Figure 7: CDF of the end–to–end delay for a pipeline with 4 stages. The execution times are distributed as in Figure 4; the server period is equal to the period of stage 0 and is $T^s = 8$ ms; the maximum budgets are $Q_0^s = 2.5$ ms, $Q_1^s = 2$ ms, $Q_2^s = 4.5$ ms, and $Q_3^s = 6.5$ ms.

[4] L. Abeni, N. Manica, and L. Palopoli. Efficient and robust probabilistic guarantees for real-time tasks. *Journal of Systems and Software*, 85(5):1147 – 1156, 2012.

[5] G. Bernat, A. Colin, and S. Petters. pwcet: A tool for probabilistic worst-case execution time analysis of real-time systems. *REPORT-UNIVERSITY OF YORK DEPARTMENT OF COMPUTER SCIENCE YCS*, 2003.

[6] T. Cucinotta and L. Palopoli. Qos control for pipelines of tasks using multiple resources. *Computers, IEEE Transactions on*, 59(3):416–430, March 2010.

[7] L. Cucu and E. Tovar. A framework for the response time analysis of fixed-priority tasks with stochastic inter-arrival times. *ACM SIGBED Review - Special issue: The work-in-progress (WIP) session of the RTSS 2005*, 3(1):7–12, January 2006.

[8] J. L. Diaz, D. F. Garcia, K. Kim, C. G. Lee, L. Lo Bello, J. M. López, S. L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 289–300. IEEE, 2002.

[9] J. L. Diaz, J. M. López, M. Garcia, A. M. Campos, K. Kim, and L. Lo Bello. Pessimism in the stochastic analysis of real-time systems: Concept and applications. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 197–207. IEEE, 2004.

[10] D. Fontanelli, L. Palopoli, and L. Abeni. The Continuous Stream Model of Computation for Real–Time Control. In *Proceedings of the IEEE Real-Time Systems Symposium*, Vancouver, Canada, 4-6 Dec. 2013. IEEE.

[11] M. K. Gardner and J. Liu. Analyzing stochastic fixed-priority real-time systems. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS99)*, pages 44–58. Springer, March 1999.

[12] P. Jayachandran and T. Abdelzaher. A delay composition theorem for real-time pipelines. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 29–38. IEEE, 2007.

[13] P. Jayachandran and T. Abdelzaher. Delay composition algebra: A reduction-based schedulability algebra for distributed real-time systems. In *Proceedings of the Real-Time Systems Symposium (RTSS)*, pages 259–269. IEEE, 2008.

[14] P. Jayachandran and T. Abdelzaher. Delay composition in preemptive and non-preemptive real-time pipelines. *Real-Time Systems*, 40(3):290–320, 2008.

[15] D.-I. Kang, R. Gerber, and M. Sakena. Performance-based design of distributed real-time systems. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, pages 2–13, June 1997.

[16] D.-I. Kang, R. Gerber, and M. Saksena. Parametric design synthesis of distributed embedded systems. *IEEE Trans. Computers*, 49(11):1155–1169, 2000.

[17] K. Kim, J. L. Diaz, L. Lo Bello, J. M. López, C. G. Lee, and S. L. Min. An exact stochastic analysis of priority-driven periodic real-time systems and its approximations. *IEEE Transactions on Computers*, 54(11):1460–1466, 2005.

[18] N. Manica, L. Palopoli, and L. Abeni. Numerically efficient probabilistic guarantees for resource reservations. In *Emerging Technologies Factory Automation (ETFA), 2012 IEEE 17th Conference on*, pages 1–8, Sept 2012.

[19] D. Maxim and L. Cucu-Grosjean. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *Proceedings of the IEEE Real-Time Systems Symposium*, Vancouver, British Columbia, Canada, December 2013.

[20] A. Mills and J. Anderson. A stochastic framework for multiprocessor soft real-time scheduling. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 311–320. IEEE, April 2010.

[21] L. Palopoli, D. Fontanelli, L. Abeni, and B. Villalba Frias. An analytical solution for probabilistic guarantees of reservation based soft real–time systems. *Parallel and Distributed Systems, IEEE Transactions on*, PP(99):1–1, 2015.

[22] L. Palopoli, D. Fontanelli, N. Manica, and L. Abeni. An analytical bound for probabilistic deadlines. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 179–188. IEEE, 2012.