Soft Real–Time Scheduling for Embedded Control Systems ⋆

Daniele Fontanelli [a], Luca Greco [b], Luigi Palopoli [a]

[a] *Dipartimento di Scienza e Ingegneria dell'Informazione (DISI), University of Trento, Via Sommarive 14, Povo (TN), Italy*

[b] *L2S - EECI - Supélec, Université Paris Sud XI, 3 rue Joliot-Curie, 91192 - Gif sur Yvette, France*

Abstract

We deal with the following problem: how to implement a feedback controller on a CPU, with variable computation time, on a CPU that is shared with other tasks so that 1) the task satisfies its Quality of Control (QoC) requirements, 2) the CPU can be efficiently managed to host other concurrent activities. We propose a solution based on the combination of a predictable model of computation and of a scheduling algorithm that enables a fine control on the evolution of the delay. This combination of techniques enables us to set up an optimisation problem where the scheduling parameters are synthesised in order to minimise the CPU utilisation and guarantee the QoC requirements for the control task.

Key words: Adaptive algorithms, Computer controlled systems, Control engineering applications of computers, Stochastic jump processes, Time schedule controllers, Real-time tasks, Real-time systems

1 Introduction

In the industrial practise, we frequently observe a discrepancy between the performance of control systems at design time and in their final implementation. An effective strategy to address this problem is by characterising the limitations to the performance that a platform can provide (Quality of Service (QoS)), and by establishing a relation with the Quality of Control (QoC) offered by the system (stability, robustness, H_2 norms, etc.). In classical digital control design, the QoS is simply associated with the $(T,\ D)$ pair, where T denotes the sampling time and D is the loop delay. The standard tools of digital control allow the designer to identify the QoC performance *given* a $(T,\ D)$ QoS specification. The $(T,\ D)$ pair is easy to identify when the controller is implemented using dedicated hardware resources. However, in a modern system (e.g., an automobile) hardware sharing is key to both cost reduction and simplification of system engineering. In this paper, we focus on a particular hardware resource: the Central Processing Unit (CPU). Feedback controllers are implemented as concurrent tasks, which are periodically activated by a timer. Control tasks compete for the CPU with other tasks, some equally important as the control task, such as sensor data processing, communication and security tasks. A scheduler allocates the CPU in presence of simultaneous execution requests. The presence of other tasks introduces random delays and input jitter. This problem is traditionally addressed using a combination of solutions. The adoption of a time–triggered model of computation [16] allows the control task to receive input samples with a very accurate periodicity T and to release its output after a fixed delay D (deadline). The real–time scheduling theory [20] offers analytical conditions for all tasks to meet their deadlines. Thereby, it is possible to recover the $(T,\ D)$ QoS model and apply classical digital control approaches [31]. The recent technological developments are gradually restricting the space for this approach. The use of such sensors as cameras, RADARS, or laser scanners introduces a large variability in the computation time. In these cases, the hard real-time scheduling theory is overly conservative and reduces the number of applications that can be executed simultaneously. This is hardly acceptable, since the strict respect of every deadline is not needed in several control systems [5]. This new generation of applications calls for scheduling solutions combining performance guarantees with an efficient resource utilisation.

Contributions of the paper. The objective of this paper is to ensure a safe coexistence in the same CPU of

⋆ This paper was not presented at any IFAC meeting. This paper subsumes and extend the results previously reported in a conference paper by the same authors [11].

Corresponding author D. Fontanelli. Tel. +39-0461-882080. Fax +39-0461-282093.

Email addresses: fontanelli@disi.unitn.it (Daniele Fontanelli), lgreco@ieee.org (Luca Greco), palopoli@disi.unitn.it (Luigi Palopoli).

control tasks with *guaranteed QoC* with the largest possible number of concurrent activities. In the wide range of possible applications of our technique, we have isolated two significant scenarios for illustrative purposes. In the first one, we aim for the maximum number of applications that receive real–time guarantees along with the control task. In the second one, the CPU is shared with best effort activities and we aim to give the latter optimised chance of execution. The cornerstone of our approach is a generalisation of the classical (T, D) QoS model. In our model, we still have a period T used to trigger the activation of the control tasks, but samples are not always collected at the timer expiration. The loop delay D is not fixed, but it evolves according to a stochastic model $\mathcal{F}_D$. Because of the possible stochastic switches in the loop delay, the resulting closed loop dynamics is that of a Stochastic Jump Linear System [22]. The construction of this model allows us *to set up an optimisation problem* where the decision variables are the scheduling parameters of the control task, and the goal is to minimise the CPU utilisation under the constraint that the system attains some specified QoC goals. The result of this process can be different for different application scenarios. In some cases, we generate a static scheduling policy, where the scheduling parameters are fixed, in others a dynamic policy, where they are subject to change depending on the delay accumulated during the execution. The two key enablers for this approach are the adoption of an analytically tractable notion of QoC (second moment stability [9]) and of a predictable model of computation. The latter, in combination with a soft real–time scheduling algorithm, allows us to assign scheduling parameters so that delays evolve according to the model $\mathcal{F}_D$ resulting from the optimisation [30,1].

The paper is organised as follows. In Section 2, we present our model of computation and our scheduling algorithm. In Section 3, we show how to set up the control problem. In Section 4, we describe the optimisation problem resulting from the combination of the model of computation and of our QoC objective. In Section 5, we offer a numeric evaluation based on a large number of simulations. In Section 6, we compare our approach with existing work. Finally, in Section 7 we draw our conclusion and announce the future working directions.

2 Platform model

Definitions. We consider a scenario where a set of tasks $\mathcal{T} = \{\tau_i\}\ i \in \{1, \ldots, n_\tau\}$, $n_\tau \in \mathbb{Z}$ share the same computation platform. Some elements of $\mathcal{T}$ are *real–time tasks* (which include the tasks used for control purposes), the others are *best–effort tasks*. The former have temporal constraints on their execution, while the latter do not receive any kind of temporal guarantees: their execution is optional but adds value to the quality perceived by the user. A real–time task τ_i consists of a stream of jobs $J_{i,j}$, $j \in \mathbb{Z}_{\geq 0}$. Each job $J_{i,j}$ is activated at time $r_{i,j}$ and finishes at time $f_{i,j}$ after executing for a time $c_{i,j}$. Job $J_{i,j}$ is also characterised by a deadline $d_{i,j}$, that is respected if $f_{i,j} \leq d_{i,j}$ and is missed if $f_{i,j} > d_{i,j}$. We focus on *periodic* tasks with *task period* T_i (i.e. $r_{i,j+1} = r_{i,j} + T_i$), where each activation time is also the deadline of the previous instance (i.e. $d_{i,j} = r_{i,j+1}$). The task is said *hard* real–time if all deadlines are met, and is said *soft* real–time if deadlines are met with a given probability. The quantity $U_i = \sup_j \frac{c_{i,j}}{T_j}$ is defined *worst case utilisation*.

2.1 The Scheduling Algorithm

In this paper, we assume that the presence of a CPU scheduler adopting the *Resource Reservations (RR)* policy. Resource Reservations are shortly explained as follows. Each task τ_i is associated with a reservation pair (Q_i, R_i), meaning that it is allowed to execute for Q_i (*budget*) time units in every interval of length R_i (*reservation period*). Clearly, the budget Q_i has to be chosen in the range $\{0, \ldots, R_i\}$. The bandwidth allocated to the task is defined as $B_i = Q_i/R_i$ and corresponds to the fraction of CPU time allocated to the task. The reservation period R_i is typically chosen as an integer sub-multiple of the task period: $T_i = N_i R_i$, $N_i \in \mathbb{N}$.

The particular implementation of the RR approach that we consider is the Constant Bandwidth Server (CBS) [1]. When the CBS is used, the bandwidths assigned to the different tasks have to be chosen respecting the following constraint:

$$\sum_i B_i \leq U_{\text{lub}}. \tag{1}$$

with $U_{\text{lub}} = 1$. The meaning of this condition is that the sum of the bandwidth assigned to the different tasks cannot exceed a bound (U_{lub}), which quantifies the efficiency of the scheduling algorithm (100% for the CBS, if the OS overheads are neglected). We remark that this scheduling technology is currently implemented in a popular real–time patch of the Linux Kernel [1] and is likely to become mainstream in the future releases.

Choosing the Reservation Parameters. When condition (1) is respected, the CBS enjoys the *temporal isolation property*, meaning that temporal guarantees can be given to each task based on its computation requirements and on its scheduling parameters, regardless of the parameters of the other tasks in the system. Therefore, the policy used to choose the reservation parameters (Q_i, R_i) depends on the real–time constraints of the task and on its computation requirements.

A first possibility is to use a *static policy*, whereby such parameters are chosen once and for all and kept constant throughout the execution of the task. For instance, for an hard real–time task, we can guarantee the respect of all

[1] `http://gitorious.org/sched_deadline/pages/Home`

its deadlines as long as it is assigned a bandwidth greater than or equal to its worst case utilisation [1]: $B_i \geq U_i$. For a soft real–time task with known distributions of the computation time, we can compute a bound for the probability that it will meet its deadlines for any choice of the budget and of the reservation period [2].

A second possibility is to use a *dynamic policy*, where the bandwidth is changed in every job to adapt the QoS provided by the task. In principle, it is possible to change the bandwidth by operating on the budget and/or on the reservation period. However, for periodic tasks, the reservation period is set to an integer sub-multiple of the task period and adaptations are made by changing the budget Q_i. This particular choice is mandated by efficiency reasons because it reduces the waste of computing power in the scheduling process [8]. The adaptation of the budget is typically made by a feedback control loop, where measurements collected inside the OS are used to assess the QoS and to fine-tune it as required. For this reason such policies are usually referred to as *adaptive reservations* [29]. In the following we will provide a concrete example of such a policy.

As far *best effort* activities are concerned, the CBS can be complemented with additional mechanisms that enable an effective *reclaiming* of unused bandwidth. As discussed in Section 4, an effective solution of this kind can be built relying on an adaptive reservation mechanism for the real–time tasks.

2.2 QoS Model for the control task

We now focus on a specific task implementing the controller, which will be denoted by τ removing the i subscript. Our objective is now to construct a QoS model $(T, \mathcal{F}_D)$ for the control task that describes the evolution of the delays introduced in its computations for different choices of the scheduling parameters. In our framework, the control task is soft real–time and shares the CPU with the other tasks in the set $\mathcal{T}$.

The task is scheduled by a reservation (Q, R) and, in view of the temporal isolation property, evolves independently of the other tasks. For the choice of Q, we will consider dynamic policies (in which the budget Q_j changes for every job J_j) and static policies. The model below is derived for the case of dynamic policies, and the case of static policies is easily recovered by imposing the constraint $Q_j = Q_{j-1}$ for all j. The task τ is activated with period $T = NR$; each activation produces a job J_j that takes a sample y_j of the plant output and produces control value u_j. To construct a QoS model that can be used in the control design, we need to know the instant s_j when y_j is sampled (*sampling time*) and the instant v_j when u_j is released (*release time*). As shown next, this is possible by the combination of an appropriate model of computation with a CBS scheduler.

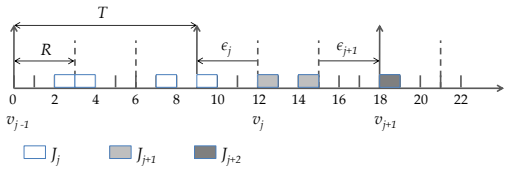

Figure 1. Example of schedule of a task ($T = 9$, $R = 3$, $N = 3$).

Model of computation. Despite the periodical activation of the task, the collection of input samples and the release of the computation result are affected by several sources of timing fluctuations (jitter) and computation time variability. To address this problem, we introduce a set of rules (a model of computation) that specify the possible sampling times s_j and the release times v_j.

Rule 1: If a job J_j finishes *before* its deadline (i.e. $f_j < d_j = r_j + T$), the release of the output u_j is deferred to the end of the period (i.e. $v_j = r_j + T$ is forced).

Rule 2: If a job J_j finishes *after* its deadline (i.e. $r_j + T + (D_j - 1)R < f_j \leq r_j + T + D_j R$ with $D_j \in \mathbb{N}$), the release of the output u_j takes place immediately at the end of the present reservation period (i.e. $v_j = r_j + T + D_j R$ is enforced and D_j is the delay of job J_j).

Rule 3: If a job J_j experiences a delay D_j greater than a threshold, it is cancelled and a new job J_{j+1} is activated.

Rule 4: The output y_j is sampled at the same time the control u_{j-1} is released (at instant $s_j = v_{j-1}$).

The purpose of Rule 1 is to reduce the output jitter due to early terminations. As an illustrative example, consider the schedule in Figure 1. The job J_{j+1} finishes at time 15 but releases its output only at time 18. Rule 2, together with Rule 1, prevents output releases that are not aligned with a reservation period. As a consequence the delay experienced by a job is always an integer multiple D_j of the reservation period. With reference to Figure 1, job J_j finishes with a delay $D_j = 1$. Rule 3 avoids the accumulation of very large delays which are impossible to be compensated. In what follows we assume that the maximum tolerated delay is one task period $T = NR$ (i.e. $\max D_j = N$). As a consequence of Rule 1, Rule 2 and Rule 3 the release time for the output of J_j is given by

$$v_j = r_j + T + D_j R = (j+1)T + D_j R, \qquad (2)$$

with $D_j \in \{0, 1, \ldots, N\}$. In view of such expression for v_j, Rule 4 ensures that output sampling always occurs at the beginning of a reservation period, even if the task can receive its budget later in the same reservation period (the job J_j in Figure 1 samples y_j at time 0, but starts at time 2).

A few remarks on this model of computation are in order. If the computation consistently terminates within the deadline, the application of the rules recovers the standard time–triggered behaviour [14], with periodic sampling and a fixed delay equal to T in the computation. On the contrary, when the task execution gets delayed, Rule 4 allows a more flexible management of the sampling instants than a time–triggered approach: when the next job starts its execution, it is given the possibility of using fresher data. For instance, job J_{j+1} in Figure 1 will use the data sampled at time 12 rather than the one sampled at time 9. One potential drawback of sampling on v_{j-1} instead of sampling on r_j is that the classical analysis based on periodic updates for y_j and u_{j-1} is not applicable. However, as discussed in the next section, this problem is easy to solve and its importance is outweighed by the advantages. In some respects, we could say that the periodic activation is used to set a periodic reference pace for the control task. However, the communication between the control task and the environment is event–triggered, and the communication events are generated the earliest instant between the periodic activation of the job and the termination of the previous one. As a final remark, this model of computation is easy to implement using the techniques proposed in the literature on time–triggered systems on top of an operating system equipped with a CBS scheduler [14,16,29]. We will not delve into these implementation details for the sake of brevity.

Evolution of the Delays. Let Q_j denote the budget allocated to job J_j. The reservation period R is held constant (and equal to an integer submultiple of the task period T) to decide the granularity of the CPU allocation (a smaller value for R corresponds to a more fluid allocation but to a greater OS overhead). The temporal evolution of the task is described by introducing a state variable, called *scheduling error*, given by the difference between the output release time v_j and the soft deadline d_j:

$$\epsilon_j \triangleq v_j - d_j = v_j - r_{j+1}. \tag{3}$$

As an example, the job J_j in Figure 1 uses four reservation periods to complete. So its output release time is $v_j = 12$ and the resulting scheduling error $\epsilon_j = 3$ time units, i.e. a reservation period R. A positive value for ϵ_j means that J_j finished beyond the deadline, i.e. it received less bandwidth than it needed. Conversely, a negative value means that it finished before its deadline (the assigned bandwidth was greater than the task needed, as happened to job J_{j+1} in Figure 1).

The dynamic evolution of the scheduling error can be found in [**?**]. The combination of such dynamics with the rules of the model of computation produces the following expression for the evolution of the delay D_j:

$$D_{j+1} \triangleq \frac{\epsilon_{j+1}}{R} = S(D_j) + \left\lceil \frac{c_{j+1}}{Q_{j+1}} \right\rceil - N, \tag{4}$$

where $S(x) \triangleq \max(\min(x, N), 0)$. The loop delays evolve in a bounded and discrete set: $D_j \in \{0, 1, \ldots, N\}$. In this model the computation time c_{j+1} (which is unknown before the execution of the job) plays the role of an exogenous disturbance term. For the case of dynamic policies, Q_j can be seen as an input variable and used to control the evolution of the delays.

3 The control problem

The control task τ is used to control a linear time–invariant and strictly proper plant. We can describe the plant by a discrete–time model obtained using the reservation period R as sampling time [2]:

$$x(k+1) = Ax(k) + Bu(k) \tag{5}$$

with $x \in \mathbb{R}^{n_x}$, $u \in \mathbb{R}^{n_u}$, $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$, $k \in \mathbb{Z}_{\geq 0}$. The control $u(k)$ is held constant between two successive release times, that is $u(k) = u_{j-1}$ for $k \in \{v_j/R, \ldots, v_{j+1}/R\}$, where u_{j-1} is the control value released at v_j. The output y_j sampled at v_j is given by $y_j = Cx(v_j/R)$ with $y \in \mathbb{R}^{n_y}$ and $C \in \mathbb{R}^{n_y \times n_x}$ output matrix of the plant (5). The task τ implements the following linear controller

$$\begin{aligned} z_{j+1} &= A_c z_j + B_c y_j \\ u_j &= C_c z_j + D_c y_j, \end{aligned} \tag{6}$$

where j is the index of the j-th job releasing its output at v_j, $z_j \in \mathbb{R}^{n_z}$, $A_c \in \mathbb{R}^{n_z \times n_z}$, $B_c \in \mathbb{R}^{n_z \times n_y}$, $C_c \in \mathbb{R}^{n_u \times n_z}$ and $D_c \in \mathbb{R}^{n_u \times n_y}$. The controller (6) does not evolve on a periodical time basis, as it is updated at each new release time instant of τ and $v_j - v_{j-1}$ is not constant for $j \in \mathbb{N}$. Nonetheless, such a controller is designed assuming *a nominal periodic behaviour* with period $T = NR$, computation delay equal to the period, and a ZoH semantics. Let $F_j \in \{0, \ldots, 2N\}$ denote the integer variable describing the number of reservation periods in which the control value u_{j-1} is held constant. Recalling (2), it is immediate to see that $F_j \triangleq \frac{v_j - v_{j-1}}{R} = N + D_j - D_{j-1}$. In the nominal condition where both jobs J_{j-1} and J_j do not suffer delays we have $F_j = N$. According to this definition, the controlled system dynamics between the release time of job J_{j-1} and the release time of job J_j is given by

$$\begin{aligned} x_{j+1} &= A^{F_j} x_j + \sum_{t=0}^{F_j - 1} A^{F_j - t - 1} B u_{j-1} \\ y_j &= C x_j, \end{aligned} \tag{7}$$

[2] Indeed, according to the rules in the previous section, each output sampling and control release operation takes place at some release time v_j, which is always an integer multiple of the reservation period.

where $x_j \triangleq x\,(v_j/R)$. If the delay is greater than N, then a drop event takes place limiting the delay to N (Rule 3). The drop event can be managed either by holding the previous control value (drop and hold), or by zeroing it (drop and zero). In both cases we consider the controller state z_j to be held ($z_j = z_{j-1}$). Therefore, in the drop case, the dynamics of the controller (6) has to be modified, while the dynamics (7) remains unchanged. In order to model the constant delay (of N reservation periods) and the drop event, we introduce the state variable $\zeta_j \in \mathbb{R}^{n_u}$. In this paper, we adopt the drop and hold semantics, with the following dynamics for the controller:

$$\underbrace{\begin{aligned} z_{j+1} &= A_c z_j + B_c y_j \\ \zeta_{j+1} &= C_c z_j + D_c y_j \\ u_j &= C_c z_j + D_c y_j \end{aligned}}_{\text{no drop}} \quad \text{and} \quad \underbrace{\begin{aligned} z_{j+1} &= z_j \\ \zeta_{j+1} &= \zeta_j \\ u_j &= \zeta_j \end{aligned}}_{\text{drop and hold}}, \tag{8}$$

The drop and zero case is obtained by simply replacing $\zeta_{j+1} = \zeta_j$, $u = \zeta_j$ with $\zeta_{j+1} = u_j = 0$.

The resulting closed loop system has the following switching dynamics

$$\begin{aligned} \xi_{j+1} &= \tilde{A}_{\phi(j)} \xi_j \\ y_j &= \tilde{C} \xi_j \end{aligned} \tag{9}$$

where $\xi_j = [x_j^T, z_j^T, \zeta_j]^T \in \mathbb{R}^{n_x+n_z+n_u}$ and $\tilde{C} = [C, 0, 0] \in \mathbb{R}^{n_y \times (n_x+n_z+n_u)}$. The piecewise constant function $\phi : \mathbb{Z}_{\geq 0} \to \{0, \ldots, 3N+1\}$ rules the switchings among the different subsystems according to the delay evolution (4) and the drop policy. Indeed, we have $2N+1$ possible values of F_j for the case of regular evolution (no drop event, i.e. $\phi(j) = 0, \ldots, 2N$), and each of them generates a possible closed loop dynamics. Additionally, we have to account for $N+1$ dynamics stemming from a drop event (i.e. $\phi(j) = 2N+1, \ldots, 3N+1$). The expression of matrices $\tilde{A}_{\phi(j)}$ can be easily derived combining (7) and (8). For $\phi(j) = 0, \ldots, 2N$, we get $2N+1$ closed loop matrices for the regular dynamics

$$\tilde{A}_{\phi(j)} = \begin{bmatrix} A^{F_j} & 0 & \tilde{B}_{\phi(j)} \\ B_c C & A_c & 0 \\ D_c C & C_c & 0 \end{bmatrix}, \tag{10}$$

with $\tilde{B}_{\phi(j)} = \sum_{t=0}^{F_j-1} A^{F_j-t-1} B$ and $F_j = 0, \ldots, 2N$. Additionally, we have $N+1$ dynamics after a drop and hold event given by indexes $\phi(j) = 2N+1, \ldots, 3N+1$ that generate

$$\tilde{A}_{\phi(j)} = \tilde{A}^{\mathrm{dh}}_{\phi(j)} = \begin{bmatrix} A^{F_j} & 0 & \tilde{B}_{\phi(j)} \\ 0 & I & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{11}$$

with $F_j = N, \ldots, 2N$ and the $\tilde{B}_{\phi(j)}$ is formally equal to the one given previously. The drop and zero matrix $\tilde{A}^{\mathrm{dz}}_{\phi(j)}$ can be obtained by $\tilde{A}^{\mathrm{dh}}_{\phi(j)}$ zeroing the last element of the last row.

3.1 Problem Formulation

We are now able to state the following problem:

Problem 1 *Design an* adaptive reservation *such that: 1) the QoS $(T, \mathcal{F}_D)$ that it offers to the control process is sufficient to sustain a QoC specification, 2) the bandwidth devoted to the control task is minimised.*

QoC and bandwidth consumption are in evident tradeoff. Indeed, if the bandwidth devoted to the control task τ is permanently greater than the worst case utilisation, all jobs complete within their deadline [1], and the controlled system always evolves with its nominal dynamics $\tilde{A}_N$ ($F_j = N$ for all jobs J_j) where the QoC specifications are respected by design. Conversely, if the bandwidth granted to τ is small, the jobs are often dropped and the system ends up executing with its open loop dynamics ($F_j = 2N$) most of the times.

4 Stochastic Model

The sequence of the computation times of the control task τ is modelled as an independent and identically distributed (i.i.d.), real–valued, continuous stochastic process denoted as $\{c_j\}_{j \in \mathbb{Z}_{\geq 0}}$. The random variable c_j takes values in the set $L_c = \{\underline{c}, \ldots, \overline{c}\}$, where $\overline{c}$ is the worst case execution time (WCET) and $\underline{c}$ is the best case execution time (BCET) and is distributed according to the probability density function (pdf) $f(c_j)$. From (4), it follows that $\{D_j\}_{j \in \mathbb{Z}_{\geq 0}}$ is itself a stochastic process that is related to the process $\{c_j\}_{j \in \mathbb{Z}_{\geq 0}}$, the budget Q_j and the value of the previous delay D_{j-1}. Contrary to $\{c_j\}_{j \in \mathbb{Z}_{\geq 0}}$ this process is discrete valued. In the QoS model, $\mathcal{F}_D$ is defined by the set of all possible values of the delay ($\{0, \ldots, N\}$) and by the associated probability.

In order to attain the desired QoC goals, we look for a time invariant dynamic policy for the QoS $(T, \mathcal{F}_D)$, with the budget chosen as a function of the delay experienced in the previous job $Q_j = Q\{D_{j-1}\}$ (which as a special case can be constant). With this choice and assuming the process $\{c_j\}_{j \in \mathbb{Z}_{\geq 0}}$ stationary (it is indeed i.i.d.), the process describing the evolution of

D_j and also the drop events is a finite-state homogeneous, discrete-time Markov chain (FSH MC). Denote by $\{\sigma(j)\}_{j\in\mathbb{Z}_{\geq 0}}$ such a MC, which takes on values in the set $L_\sigma = \{0,\dots,N+1\}$, with the meaning that if $\sigma(j) = q\ \forall q \in \{0,\dots,N\}$ then $D_j = q$, if $\sigma(j) = N+1$ then $D_j = N$ and a drop event takes place. The stochastic characterization of $\{\sigma(j)\}_{j\in\mathbb{Z}_{\geq 0}}$ is given by the transition probability matrix $P = (p_{qg})_{(N+2)\times(N+2)}$, $p_{qg} \triangleq \Pr\{\sigma(j+1) = g \mid \sigma(j) = q\}$ and by the initial probability measure $\pi_\sigma(0) \in \mathcal{S}^{N+1}$, where $\mathcal{S}^{N+1} \triangleq \left\{\varsigma = (\varsigma_1,\dots,\varsigma_{N+2}) \in [0,1]^{N+2} \mid \sum_{i=1}^{N+2}\varsigma_i = 1\right\}$ is the $(N+1)$-dimensional canonical stochastic simplex. The evolution of the probability distribution $\pi_\sigma(j)$ of the MC σ is then given by $\pi_\sigma(j+1) = \pi_\sigma(j)P$.

Recalling (4), for every q and g such that $0 \leq q \leq N$ and $1 \leq g \leq N$, we have $p_{qg} = \Pr\left\{g = q + \left\lceil \frac{c_{j+1}}{Q\{q\}} \right\rceil - N\right\}$ and $p_{qg} = \Pr\{c_{j+1} \in V(q,g)\} = \int_{V(q,g)} f(x)dx$, where the set $V(q,g) \triangleq (\underline{a}_{qg}, \overline{a}_{qg}]$, and the values $\underline{a}_{qg} \triangleq \min\left(Q\{q\}(g-q+N-1), \bar{c}\right)$ and $\overline{a}_{qg} \triangleq \min\left(Q\{q\}(g-q+N), \bar{c}\right)$.

A special case is related to the probability to finish with a state $g = 0$ associated with the event $D_{j+1} = 0$ (i.e., the control task terminates before the deadline). In this case, due to the presence of the $S(\cdot)$ operator, we can compute $V(q,0) \triangleq (0,\ \min\left(Q\{q\}(N-q), \bar{c}\right)]$. From this expression, we can easily see that by increasing the budget $Q\{q\}$ we can increase the probability of meeting the deadline, whilst a large accumulated delay q plays adversely. Another important special case is related to the state $g = N+1$ (corresponding to a drop event), with the probability $p_{q\,N+1}$ given by: $p_{q\,N+1} = \Pr\left\{q + \left\lceil \frac{c_{j+1}}{Q\{q\}} \right\rceil - N > N\right\}$, resulting into $V(q,N+1) \triangleq \left(\min\left(Q\{q\}(2N-q),\ \bar{c}\right),\ \bar{c}\right]$ for $0 \leq q \leq N$. The set $V(q,N+1)$ can be empty if $\min\left(Q\{q\}(2N-q),\ \bar{c}\right) = \bar{c}$.

An important question is ascertaining whether the FSH MC σ is also irreducible and aperiodic, that is if there exists a unique invariant probability distribution (i.p.d.) $\bar{\pi}_\sigma$ corresponding to the steady-state probability distribution of the MC (i.e. $\lim_{h\to\infty}\pi_\sigma(h) = \bar{\pi}_\sigma$ for any $\pi_\sigma(0)$) [22]. In such a case the MC σ will be denoted by FSHIA MC. Conditions for this are stated in the following.

Lemma 1 *The FSH MC σ is irreducible and aperiodic if*

$$\begin{aligned} &0 < Q\{q\} < \left\lfloor \frac{\bar{c}}{N} \right\rfloor,\ \forall q \in \{0,\dots,N\}, \\ &0 < Q\{N+1\} < \left\lfloor \frac{\bar{c}}{N-1} \right\rfloor. \end{aligned} \tag{12}$$

Proof. It can be easily verified that for σ to be a FSHIA MC it is sufficient that all elements on the diagonal, on the first subdiagonal and on the first superdiagonal of the transition matrix P be non zero. Indeed, a tridiagonal non-negative matrix P with no zero elements on its diagonals is such that $P^m > 0$ for $m > n-1$ with n the dimension of P. This condition (called Frobenius' test for primitivity [25]) requires, in our case, that the integration domains $V(q,q-1)$, $V(q,q)$ and $V(q,q+1)$ related to the entries on the three main diagonals of P be not empty, and that the pdf $f(\cdot)$ be not identically null on them. The latter requirement can be satisfied if $f(\cdot)$ is such that $\int_{c-1}^{c} f(x)dx \neq 0\ \forall c \in \{\underline{c},\dots,\bar{c}\}$. To satisfy the former requirement we impose a lower bound on $V(q,q+1)$ and an upper bound on $V(q,q-1)$. That is $\min\left(Q\{q\}N, \bar{c}\right) < \bar{c}$ and $\min\left(Q\{q\}(N-1), \bar{c}\right) > 0$. Such conditions can be easily reformulated as in (12). □

Stochastic Description of the Switching Process. In this section we derive a stochastic description for the process $\{\phi(j)\}_{j\in\mathbb{Z}_{\geq 0}}$ ruling the switchings of the closed loop system (9), which is thus a Stochastic Jump Linear System (SJLS). The process $\{\phi(j)\}_{j\in\mathbb{Z}_{\geq 0}}$ takes values in the set $L_\phi = \{0,\dots,3N+1\}$ as follows:

(1) $\phi(j) = N - \sigma(j-1) + \sigma(j)$ for $\sigma(j-1), \sigma(j) < N+1$;
(2) $\phi(j) = 2N - \sigma(j-1)$ for $\sigma(j-1) < N+1$ and $\sigma(j) = N+1$;
(3) $\phi(j) = 2N + 1 + \sigma(j)$ for $\sigma(j-1) = N+1$ and $\sigma(j) < N+1$;
(4) $\phi(j) = 3N+1$ for $\sigma(j-1), \sigma(j) = N+1$.

According to this definition, we can define for each $i \in L_\phi$ a set of pair(s) $O_i = \{(a,b) \in L_\sigma \times L_\sigma \mid \sigma(j-1) = a,\ \sigma(j) = b \text{ s.t. } \phi(j) = i\}$. Hence we can write

$$\pi_{\phi_i}(j) = \Pr\{\phi(j) = i\} = \sum_{(a,b)\in O_i} \Pr\{\sigma(j) = b, \sigma(j-1) = a\}. \tag{13}$$

In order to provide the stochastic characterization of ϕ, we define another process describing the evolution of two consecutive steps of the MC σ: $\hat{\sigma}(j) = (\sigma(j), \sigma(j-1))$, taking values in the set $L_{\hat{\sigma}} = L_\sigma \times L_\sigma$. Hence we have

$$\begin{aligned} \hat{\pi}_{ab}(j) &\triangleq \Pr\{\hat{\sigma}(j) = (a,b)\} = \Pr\{\sigma(j) = b,\ \sigma(j-1) = a\} \\ &= \Pr\{\sigma(j) = b \mid \sigma(j-1) = a\}\Pr\{\sigma(j-1) = a\} \\ &= p_{ab}\pi_{\sigma_a}(j-1). \end{aligned}$$

Recalling the definition of π_σ we have $\pi_{\sigma_a}(j-1) = \sum_{c=0}^{N+1} p_{ca}\pi_{\sigma_c}(j-2) = \sum_{c=0}^{N+1}\hat{\pi}_{ca}(j-1)$, and

$$\hat{\pi}_{ab}(j) = p_{ab}\sum_{c=0}^{N+1}\hat{\pi}_{ca}(j-1) = \hat{\pi}(j-1)v_a p_{ab}, \tag{14}$$

with $\hat{\pi} \triangleq [\hat{\pi}_{00}, \hat{\pi}_{01}, \dots, \hat{\pi}_{N+1,N}, \hat{\pi}_{N+1,N+1}]$ and v_a is the a-th column of the matrix $V \in \{0,1\}^{(N+2)^2\times(N+2)}$ given

by $V = [I_{N+2}, \ldots, I_{N+2}]^T$. Equation (14) can be written as

$$\hat{\pi}(j) = \hat{\pi}(j-1)\hat{P}, \tag{15}$$

with $\hat{P} \triangleq [v_0 p_{00}, v_0 p_{01}, \ldots, v_{N+1} p_{N+1,N+1}]$, thus revealing that $\hat{\sigma}$ is a FSH MC. A relevant issue is establishing if $\hat{\sigma}$ admits a unique i.p.d.. A potential problem is that $\hat{\sigma}$ is not always irreducible even if σ is irreducible, which could lead to zero steady state probability for some initial state of $\hat{\sigma}$. However, the following proposition holds.

Proposition 1 *[10] If σ is a FSHIA MC with unique steady state distribution $\bar{\pi}_\sigma = [\bar{\pi}_{\sigma_0}, \bar{\pi}_{\sigma_1}, \ldots, \bar{\pi}_{\sigma_{N+1}}]$ and transition probability matrix $P = (p_{ab})_{(N+2)\times(N+2)}$, then $\hat{\sigma}$ is a FSH MC with a unique steady state distribution $\hat{\bar{\pi}} = [\hat{\bar{\pi}}_{00}, \hat{\bar{\pi}}_{01}, \ldots, \hat{\bar{\pi}}_{N+1,N}, \hat{\bar{\pi}}_{N+1,N+1}]$ with $\hat{\bar{\pi}}_{ab} = p_{ab}\bar{\pi}_{\sigma_a}$.*

The process $\{\phi(j)\}_{j\in\mathbb{Z}_{\geq 0}}$ is not a FSH MC, but its distribution $\pi_\phi(j)$ is linearly related to the one of the FSH MC $\hat{\sigma}$. Indeed, recalling (13) we can write

$$\pi_\phi(j) = \hat{\pi}(j)W, \tag{16}$$

for a suitable matrix $W = (w_{lh})_{(N+2)^2\times(3N+2)}$, with $w_{lh} \in \{0, 1\}$. Hence, the process $\{\phi(j)\}_{j\in\mathbb{Z}_{\geq 0}}$ admits a unique steady state distribution $\bar{\pi}_\phi$ which, recalling Proposition 1, is given by

$$\bar{\pi}_\phi = \hat{\bar{\pi}}W, \tag{17}$$

where $\hat{\bar{\pi}}$ is the steady state probability of the FSH MC $\hat{\sigma}$.

4.1 The notion of QoC

The description of process $\{\phi(j)\}_{j\in\mathbb{Z}_{\geq 0}}$ allows us to establish a link between the QoS model $(T, \mathcal{F}_D)$ and the QoC. We use the system stability as our notion of QoC. To this end, we adapt the definitions of Second Moment stability [9] to our class of stochastic processes, namely processes which are not FSH MC, but whose distributions are linearly related to the distribution of a FSH MC (see (16)).

Definition 1 *Let us consider the stochastic process $\{\sigma(t), \phi(t)\}_{t\in\mathbb{Z}_{\geq 0}}$ taking values in the set $L_\sigma \times L_\phi \triangleq \{1, \ldots, n\} \times \{1, \ldots, m\}$ with $n \geq m \geq 1$. Assume that $\{\sigma(t)\}_{t\in\mathbb{Z}_{\geq 0}}$ is a FSH MC with transition probability matrix $P = (p_{li})_{n\times n}$ and initial distribution $\pi_\sigma(0) = \pi_{\sigma 0}$, and $\{\phi(t)\}_{t\in\mathbb{Z}_{\geq 0}}$ has probability distribution given by $\pi_\phi(t) = \pi_\sigma(t)\bar{W}$ with $W = (w_{lj})_{n\times m}$ a row stochastic matrix. The SJLS*

$$x_{t+1} = A_{\phi(t)}x_t \tag{18}$$

with $x \in \mathbb{R}^M$, $A_i \in \mathbb{R}^{M\times M}$, $i \in L_\phi$ is said second moment stable *(SM-stable) with respect to $\Phi \triangleq \left\{(\pi_{\sigma 0}, \pi_{\sigma 0}W) \in \mathcal{S}^{n+m-2} \mid \pi_{\sigma 0} \in \mathcal{S}^{n-1}\right\}$ if for any initial condition $x_0 \in \mathbb{R}^M$ and any initial distribution $(\pi_{\sigma 0}, \pi_{\phi 0}) \in \Phi$ we have: $\lim_{t\to\infty} \mathrm{E}\left\{\|x_t(x_0)\|^2\right\} = 0$, where $x_t(x_0)$ is a sample solution of (18) with initial condition x_0.*

For SM-stability we can use the following sufficient condition [13].

Theorem 2 *If there exist n matrices $G_l = G_l^T > 0$ such that $\sum_{j=1}^m w_{lj} A_j^T \hat{G}_l A_j - G_l < 0$, $\forall l \in L_\sigma$ with $\hat{G}_l \triangleq \sum_{i=1}^n p_{li} G_i$, then the SJLS (18) is SM-stable.*

It is worth noting that, if the process $\{\phi(t)\}_{t\in\mathbb{Z}_{\geq 0}}$ is a FSH MC, then the previous condition coincides with the set of linear matrix inequalities provided in [26].

4.2 Optimal Reservation Policies

We are now in condition to solve Problem 1. Let us define the stochastic process $\{q(j)\}_{j\in\mathbb{Z}_{\geq 0}}$ as the sequence of budgets assigned to the control task τ by the scheduler. Such a process takes values in the set $\{Q\{0\}, \ldots, Q\{N+1\}\}$ representing all the possible state dependent budget values. All these values can be stacked in the optimisation vector $Q \triangleq [Q\{0\}, \ldots, Q\{N+1\}]^T$. The solution set of the optimisation problem is given by the constraints (12) on the values of the budget and by the SM-stability condition of Theorem 2. Assuming that a fraction of each reservation period R has to be reserved for the execution of other tasks than τ, then each $Q\{q\}$ must be less than a pre-specified $Q_{\max}$. Therefore, defining the vectors $\underline{Q} \triangleq [0, \ldots, 0]^T$ and $\overline{Q} \triangleq [\alpha, \ldots, \alpha, \beta]^T$, where $\alpha \triangleq \min\left(\left\lfloor \frac{\bar{c}}{N} \right\rfloor, Q_{\max}\right)$ and $\beta \triangleq \min\left(\left\lfloor \frac{\bar{c}}{N-1} \right\rfloor, Q_{\max}\right)$, the budget is constrained by: $\underline{Q} < Q < \overline{Q}$. Problem 1 can be formalised by the following nonlinear optimisation program, the *Optimal Reservation Policy - SM-Stability (ORP-SM)* problem in the vector Q and the $(N+2)^2$ matrices G_l:

ORP-SM Problem

$$\begin{aligned}
&\min_{Q} g(\bar{\pi}_\sigma, Q) \text{ s.t.} \\
&\quad \underline{Q} < Q < \overline{Q} \\
&\quad G_l = G_l^T > 0 \\
&\quad \textstyle\sum_{h=0}^{3N+1} w_{lh} \tilde{A}_h^T \hat{G}_l \tilde{A}_h - G_l < 0, \qquad \forall l \in L_{\hat{\sigma}} \\
&\quad \hat{G}_l = \textstyle\sum_{i=0}^{(N+2)^2-1} \hat{p}_{li}(Q) G_i \\
&\quad \hat{P}(Q) = (\hat{p}_{li}(Q))_{(N+2)^2\times(N+2)^2} \text{ as in (15)} \\
&\quad W = (w_{lh})_{(N+2)^2\times(3N+2)} \text{ as in (16)} .
\end{aligned}$$

The computation budget is usually chosen as an integer multiple of a time interval (which is dictated by the granularity of the clock used in the OS). For this reason the problem ORP-SM is essentially a mixed integer optimisation problem which can, in principle, be solved by exaustively searching on all the possible combination of discrete values for the vector of budget Q. For a given choice of the elements of the vector Q, the problem reduces to an LMI feasibility check, which is convex and can be solved very efficiently.

4.3 Optimisation goals

The ORP-SM solution depends on the cost index $g(\bar{\pi}_\sigma, Q)$, which represents the bandwidth minimisation criterion according to the second point of Problem 1. For a real–time platform, we identify two main problems: 1) given the control task set $\mathcal{T} = \{\tau_i\}$, with $i \in \{1, \ldots, n_\tau\}$, maximize the number of tasks n_τ that can be hosted on the platform; 2) maximize the amount of CPU time allotted to other best effort (non critical) activities sharing the platform.

4.3.1 Maximise n_τ

In order to maximize the number of control tasks that can be executed on the platform, we need to minimize the *maximum* bandwidth that each one uses. This way we can comply with the schedulability constraint (1) for a larger number n_τ. In formal terms, we can optimise w.r.t. the following cost function $g(\bar{\pi}_\sigma, Q) = \|Q\|_\infty$. When the purpose is to minimise the maximum bandwidth, dynamic adaptation of the bandwidth is immaterial, hence a *static* scheduling policy is sufficient. Therefore, the vector Q of decision variables reduces to a scalar.

4.3.2 Optimise for best effort tasks

For best effort tasks, our goal is to maximise the chance of execution whenever possible, i.e. whenever the processor is not utilised by tasks receiving real–time guarantees (first and foremost the control task). To this end, we need a reclaiming mechanism operating hand-in-hand with the CBS scheduler. There are several solutions available in the literature. The one that we advocate here lies in the track opened by Cucinotta et al. [7]. The solution is compounded of an adaptive mechanism in the scheduler (an adaptive reservation) and of an additional mechanism to redistribute the unused bandwidth to the best effort activities. The purpose of the adaptive reservation is to track the computing requirements of the control task (and of other soft real–time tasks) and to allocate it a bandwidth very close to its actual needs.

An adaptive scheduler of this kind can be obtained, in our setting, by solving the ORP-SM problem with a cost index that minimises the mean bandwidth allocated to the control task. A suitable cost index, which provides a stochastic average of the budget provided by the scheduler, is the long-run expected value of the budget itself. Assuming σ is a FSHIA MC, the cost index can be written as $g(\bar{\pi}_\sigma, Q) = \lim_{j\to\infty} \mathrm{E}\{q(j)\} = \bar{\pi}_\sigma(Q)Q$. By definition $\bar{\pi}_\sigma$ is a function of the budget Q, hence this is a nonlinear cost index.

The solution of the optimisation problem produces in this case a *dynamic* scheduling policy, where the budget is adjusted for each job based on the delay measured in the OS. Clearly, the optimisation of the average bandwidth comes at the price of a possible growth of the maximal bandwidth required by the task. This can reduce the number of tasks requiring temporal guarantees (i.e. a minimum availability of bandwidth).

5 Simulation Results

This section reports simulations that show the effectiveness of the proposed approach in the two scenarios discussed earlier. In all the experiments reported below, we synthesised a Linear Quadratic Gaussian (LQG) controller, using the same weights and assuming a nominal condition with a constant delay of one sampling period (see Rule 1). The computation model presented in Section 2.2 is adopted. In the different experiments, we have used computation times given by i.i.d. stochastic processes with three different distributions: a uniform distribution (UD), an exponential distribution (ED) and a beta distribution (BD).

Hosting multiple tasks with real–time guarantees. In the first set of experiments, we have considered a platform hosting three control tasks $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ controlling three open–loop unstable, observable and controllable plants. The task periods have been fixed to $T_1 = 10$ ms, $T_2 = 2$ ms and $T_3 = 18$ ms, each one being four times the respective reservation periods R_i. The BCETs have been chosen as $\underline{c}_1 = 125\mu$s, $\underline{c}_2 = 25\mu$s and $\underline{c}_3 = 225\mu$s. The WCETs have been fixed to a ratio of the reservation periods: $\bar{c}_i = \frac{T_i+R_i}{3} = \frac{5}{3}R_i$.

By using the classic approach the three tasks cannot be executed sharing a single real–time platform. Indeed, the strict respect of every deadline requires an allocation of bandwidth equal to the worst case utilisation [1] ($B_i = \frac{\bar{c}_i}{T_i}$), and this assignment violates the schedulability constraint in (1) ($\sum_i \frac{\bar{c}_i}{T_i} = \frac{15}{12} > 1$). However, by exploiting the knowledge on the distribution of the computation times, this limit has been overcome. For example, we have chosen the UD and BD defined in the range given by the BCET and WCET, while the ED has been truncated (and re-normalised) in the same range. The mean values μ_i and standard deviations σ_i have been chosen as follows: $\mu_1 = 1.37$ ms, $\mu_2 = 0.27$ ms, $\mu_3 = 2.46$ ms, $\sigma_1 = 0.67$ ms, $\sigma_2 = 0.13$ ms and $\sigma_3 = 1.22$ ms for

the BD; $\mu_1 = 1.53$ ms, $\mu_2 = 0.31$ ms, $\mu_3 = 2.75$ ms, $\sigma_1 = 1.06$ ms, $\sigma_2 = 0.21$ ms and $\sigma_3 = 1.9$ ms for the ED. The mean values of UD are $\mu_1 = 2.15$ ms, $\mu_2 = 0.43$ ms and $\mu_3 = 3.86$ ms. By solving the ORP-SM problem assuming a *static* allocation of the bandwidth minimizing the maximum bandwidth for each task in isolation (see Section 4.3.1), the bandwidths allocated to the three tasks were respectively: $B_1 = B_2 = 0.3$ and $B_3 = 0.35$ for UD, $B_1 = B_2 = 0.2$ and $B_3 = 0.3$ for ED, and $B_1 = B_2 = 0.2$ and $B_3 = 0.25$ for BD. As it may be expected, the UD is the distribution that required the highest allocation of bandwidth, since the shape of the distribution is the less favorable. Since the sum of the bandwidths for the three tasks was less than one for any possible combination of the distributions, it was possible to host the three tasks on the same CPU.

The previous simulations show that our approach allows us to schedule a control task even in the face of a computation time greater than the period (utilization greater than one), which is clearly out of the question in a hard real–time setting. To investigate further on this issue, we have devised a specific experiment where our scheduling proposal is confronted with a naive approach where the task is allowed to execute until the deadline with full bandwidth and, if the execution is not finished, the job is simply dropped (*drop–out* policy). To this end, we designed a linear stabilising controller for a Furuta pendulum with zero offset [12], whose linearised dynamic is represented by the continuous-time transfer function $G(s) = \frac{7.435}{s(s^2+34.63)}$ between the input torque τ and the output arm angle α. The task period was fixed to $T = 10$ ms (reservation period $R = T/6 = 1.66$ ms). The computation times were given by a i.i.d. process with uniform distribution defined in the range between BCET $\underline{c} = 1.66$ ms and WCET $\bar{c} = 11.66$ ms. In this situation, the single task was not hard real–time schedulable even if 100% of the computation time were granted. We have carried out simulations using the drop–out policy outlined above and compared it with the *static* reservation policy (Section 4.3.1). To show the generality of our approach, we have considered here a problem of tracking a piecewise constant reference, which can be easily re-cast into a stabilization problem. The reference signal was a square wave of period 10 s, $\pi/4$ amplitude and duty cycle of 50%. Figure 2 reports the logarithm of $E\{\|x(k)\|^2\}$ on 200 runs for both the *drop-out* (dashed) and the *static* (solid) reservation policy. The evident convergence of the mean squared tracking error is a direct consequence of the SM-stability. Interestingly the *drop-out* policy leaded to instability. As a final remark, even if large fluctuations of the state variables cannot be ruled out under SM-stability, in these experiments the transient behavior was always satisfactory, i.e., the deviation of the system behavior with respect to the ideal execution (where the control data is always delivered on the deadline) was consistently below 0.1 rad for all simulations run.

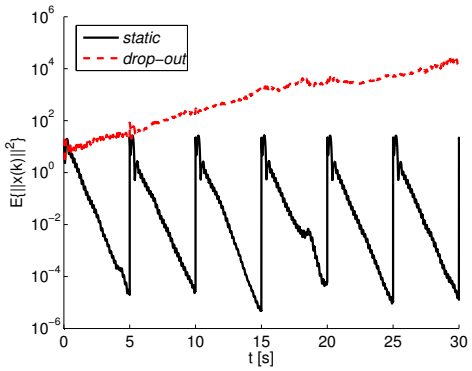

Figure 2. $E\{\|x(k)\|^2\}$ of the Furuta pendulum in case of the *static* reservation policy (solid) and *drop-out* policy (dashed). The mean is computed over 200 independent runs with random initial conditions.

Best-Effort Activities. In this set of experiments 100 unstable plants, whose order ranges between two and six, have been generated. The task period has been fixed to $T = 10$ ms, which is four times the reservation period $R = 2.5$ ms. For the computation times, the UD and BD are defined in the range [2.5, 10] ms (i.e. BCET of 2.5 ms and WCET of $T = 10$ ms), while the ED is truncated (and re–normalised) in the same range. The mean value is $\mu = 4.86$ ms (standard deviation $\sigma = 1.25$ ms) for the BD and $\mu = 5.62$ ms ($\sigma = 2.1$ ms) for the ED. We draw a comparison between a hard real–time policy (*HRT*) and the two proposed policies solution of the ORP-SM problem: the *static* policy in Section 4.3.1 and the *dynamic* policy in Section 4.3.2. Note that the with *HRT* policy the 100% of the CPU utilisation would be required. The objective of this set of experiments is to quantify the amount of bandwidth saving. To this aim, for each experiment run we measure the ratio between the average allocated budget (i.e. total budget over number of jobs) and the budget required by the *HRT* (which is constant). In Figure 3, we report the experimental probability of this ratio in form of Cumulative Density Functions (CDFs). In plain words, for a given value x of the ratio, the plot shows the experimental probability of using less than x% of the *HRT* budget. This set of experiments underscores the impact of the distributions of computation times on the QoS improvement, the UD being the most challenging and the BD the most favourable. The *dynamic* policy is the most effective in exploiting the knowledge of the distribution, e.g. for the BD the bandwidth savings are closely concentrated around 50%.

6 Related Work and Discussion

This paper owes ideas and inspiration to several pieces of existing work. One of the first known proposals that relate the real–time scheduling choice with control performance is due to Seto et. al [31]. The authors consider a set of periodic tasks that implement control loops; under

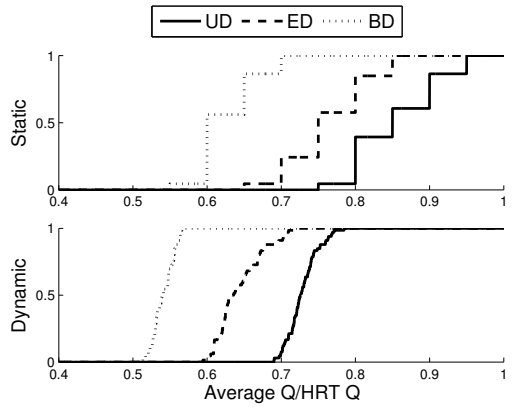

Figure 3. CDFs representing the experimental probability of using a fraction of the budget allocated to *HRT* for both *static* and *dynamic* policies.

the assumption of an exponential dependence between QoC and sampling period, they identify an analytical form for the optimal periods. An interesting evolution is a paper by Cervin et al. [4], where a feedback scheduler adapts the sampling frequency of the tasks assuming different types of analytical dependence between QoC and frequency. More recently, Buttazzo et al. [3] propose to re-modulate the sampling periods when the system undergoes an overload condition. Our work differs from these in two essential aspects: 1) the scheduling mechanism, which in our case adapts the bandwidth (leaving the period unchanged), 2) the evaluation of the QoC, which in our case is rooted in system theoretic properties.

The problem of jitter in control has been addressed by several authors. Martí et al. [23] use a monitoring mechanism that detects the jitter online and compensates for it in the control algorithm. Other authors adapt the classic notion of stability margin to make a given design robust against the timing variations related to the jitter [6,15]. In our proposal, the effect of jitter is reduced through a time–triggered model of computation [14,16]. The use of the CBS as a scheduling solution allows us to model the evolution of the delays by a Markov chain and hence construct a Stochastic Jump Linear System (SJLS), which can be easily studied in terms of QoC. A closer link can be established with the approach underlying the Jitterbug tool, proposed by Lincoln et. al [17]. If the designer provides a description of the timing behaviour of the plant along with the control algorithm, the tool evaluates the impact of the jitter on the QoC (the expected value of a quadratic cost function). The tool is based on previous results on the application Markov Jump Linear Systems to network control [27]. A tool like this could in principle be plugged into our method to encompass different notions of QoC. This requires addressing two issues: 1) the construction of a timing model that adapts the model described in Section 2 to the one requested by Jitterbug, 2) an adaptation of Jitterbug to address more general Stochastic Jump Linear Systems. The advantage of using soft real–time scheduling approaches for control has been shown in several experimental and field studies [28]. Closely related to our idea is the approach developed by Chantem et al. [19]. The authors take to an interesting development their previous work [18], by considering a (n, m) firm real–time tasking model: in each window of n tasks at least m complete within their deadline. Given a QoC optimisation problem, they derive a Markov chain describing the evolution of the (n, m) model and propose a heuristic scheduling algorithm that approximates this evolution. Our approach differs for the following reasons. First, we do not consider an all-or-nothing model of execution, where a job is forced to terminate within the deadline or dropped and allow for a delayed termination. Second, our scheduling algorithm determines an evolution of the delays which is exactly captured by the $\mathcal{F}_D$ model, and is not a heuristic approximation.

An important class of approaches that is worth a mention are state–triggered or self triggered control schemes [32,24,33], in which the idea of periodic sampling is completely replaced by other mechanisms related to the evolution of the plant. The difference with our work is remarkable, since we have a period T used to set the "reference" pace for the execution of the control. Samples can occasionally be collected upon the occurrence of *internal events* (the output release time of a job) and do not depend on the evolution of the plant.

Finally, an important source of inspiration for our work has been the literature of feedback scheduling. The idea has been pioneered by Lu et al. [21], who propose an adaptation of the task deadline based on monitoring the workload in the system. The authors assume an EDF algorithm but do not rely on a precise modelling of the system evolution. For our purposes the adaptive reservations are a more appropriate technology for the greater degree of predictability of their evolution and for the existence of reliable implementations [29].

7 Conclusion

In this paper, we have considered the problem of scheduling a control task. Starting from an assigned control law and from a stochastic description of its execution time, we derive an adaptive scheduling policy that allows us to attain stability for the system and to minimise the consumption of computation resources. Our paper is, in our evaluation, the first step in a promising direction. There are several issues reserved for future investigation.

One of the most interesting is to extend our approach to different notions of QoC than second moment stability. Another potentially important development is to develop convex relaxation of the ORP-SM problem that makes for its efficient solution.

Acknowledgements

This work was partially supported by the HYCON2 NoE, under grant agreement FP7-ICT-257462.

References

[1] L. Abeni and G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In *Proc. IEEE Real-Time Systems Symposium*, pages 4–13, Dec. 1998.

[2] L. Abeni and G. C. Buttazzo. QoS guarantee using probabilistic deadlines. In *Proceeding of the 11th Euromicro Conference on Real-Time Systems (ECRTS 1999)*, pages 242–249, York, England, UK, 9-11 June 1999.

[3] Giorgio Buttazzo, Manel Velasco, and Pau Marti. Quality-of-Control Management in Overloaded Real-Time Systems. *IEEE Trans. on Computers*, 56(2):253–266, Feb. 2007.

[4] A. Cervin, J. Eker, B. Bernhardsson, and K.E. Årzén. Feedback–feedforward scheduling of control tasks. *Real-Time Systems*, 23(1):25–53, 2002.

[5] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Arzen. How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime. *IEEE Control Systems Magazine*, 23(3):16–30, June 2003.

[6] A. Cervin, B. Lincoln, J. Eker, K.E. Arzen, and G. Buttazzo. The jitter margin and its application in the design of real-time control systems. In *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*. Gothenburg, Sweden, 2004.

[7] T. Cucinotta, L. Abeni, L. Palopoli, and G. Lipari. A robust mechanism for adaptive scheduling of multimedia applications. *ACM Transactions on Embedded Computing Systems (TECS)*, 10(4):46, 2011.

[8] T. Cucinotta, F. Checconi, L. Abeni, and L. Palopoli. Self-tuning schedulers for legacy real-time applications. In *Proceedings of the 5th European Conference on Computer Systems*, pages 55–68. ACM, 2010.

[9] Y. Fang and K.A. Loparo. Stochastic Stability of Jump Linear Systems. *IEEE Trans. on Automatic Control*, 47(7):1024–1028, August 2002.

[10] D. Fontanelli, L. Greco, and L. Palopoli. Adaptive reservations for feedback control. In *Proceedings of the 49th IEEE Conference on Decision and Control, CDC 2010*, pages 4236–4243, Atlanta, GE, USA, December 2010. IEEE.

[11] D. Fontanelli, L. Palopoli, and L. Greco. Deterministic and stochastic qos provision for real-time control systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pages 103–112, april 2011.

[12] K. Furuta, M. Yamakita, and S. Kobayashi. Swing-up control of inverted pendulum using pseudo-state feedback. *Proceedings of the Institution of Mechanical Engineers. Pt.I. Journal of Systems and Control Engineering*, 206(I4):263–269, 1992.

[13] L. Greco, A. Chaillet, and E. Panteley. Robustness of stochastic discrete-time switched linear systems with application to control with shared resources. LSS, http://hal.archives-ouvertes.fr/hal-00728093, 2012. Preprint.

[14] T.A. Henzinger, B. Horowitz, and C.M. Kirsch. Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84 – 99, jan 2003.

[15] C. Kao and A. Rantzer. Stability analysis of systems with uncertain time-varying delays. *Automatica*, 43(6):959–970, June 2007.

[16] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112 – 126, jan 2003.

[17] B. Lincoln and A. Cervin. JITTERBUG: a tool for analysis of real-time control performance. In *Proceedings of the 41st IEEE Conference on Decision and Control, CDC 2002*, pages 1319–1324, Dec. 2002.

[18] Q. Ling and M.D. Lemmon. Robust performance of soft real-time networked control systems with data dropouts. In *Proceedings of the 41st IEEE Conference on Decision and Control, CDC 2002*, volume 2, pages 1225–1230, Dec. 2002.

[19] D. Liu, X.S. Hu, M.D. Lemmon, and Q. Ling. Firm real-time system scheduling based on a novel QoS constraint. *IEEE Trans. on Computers*, 55(3):320–333, March 2006.

[20] J.W.S. Liu. *Real-time systems*. Prentice Hall, 2000.

[21] C. Lu, J.A. Stankovic, S.H. Son, and G. Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems*, 23(1):85–126, 2002.

[22] M. Mariton. *Jump linear systems in automatic control*. CRC, 1990.

[23] P. Marti, J.M. Fuertes, G. Fohler, and K. Ramamritham. Jitter compensation for real-time control systems. In *Proc. IEEE Real-Time Systems Symposium*, pages 39–48, Dec. 2001.

[24] M. Mazo and P. Tabuada. Input-to-state stability of self-triggered control systems. In *Proceedings of the 48th IEEE Conference on Decision and Control, CDC 2009 held jointly with the 2009 28th Chinese Control Conference*, pages 928–933, December 2009.

[25] C.D. Meyer. *Matrix analysis and applied linear algebra*. Society for Industrial Mathematics, 2000.

[26] T. Morozan. Stabilization of some stochastic discrete-time control systems. *Stoch. Anal. Appl.*, 1(1):89–116, 1983.

[27] J. Nilsson and B. Bernhardsson. Analysis of real-time control systems with time delays. In *Proc. IEEE Conf. on Decision and Control*, volume 3, pages 3173–3178, Dec. 1996.

[28] L. Palopoli, L. Abeni, G. Buttazzo, F. Conticelli, and M. Di Natale. Real-time control system analysis: an integrated approach. In *Proc. IEEE Real-Time Systems Symposium*, pages 131–140, Orlando, FL , USA, Nov. 2000.

[29] L. Palopoli, T. Cucinotta, L. Marzario, and G. Lipari. AQuoSA - adaptive quality of service architecture. *Software: Practice and Experience*, 39(1):1–31, 2009.

[30] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Resource Kernels: A Resource-Centric Approach to Real-Time and Multimedia Systems. In *Proc. of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.

[31] D. Seto, J.P. Lehoczky, L. Sha, and K.G. Shin. On task schedulability in real-time control systems. In *rtss*, page 13. Published by the IEEE Computer Society, 1996.

[32] M. Velasco, P. Marti, and E. Bini. On Lyapunov sampling for event-driven controllers. In *Proceedings of the 48th IEEE Conference on Decision and Control, CDC 2009 held jointly with the 2009 28th Chinese Control Conference*, pages 6238–6243, December 2009.

[33] X. Wang and M. D. Lemmon. Decentralized Event-Triggered Broadcasts over Networked Control Systems. In Magnus Egerstedt and Bud Mishra, editors, *Hybrid Systems: Computation and Control, HSCC 2008*, volume 4981 of *Lecture Notes in Computer Science*, pages 674–677, St. Louis, MO, USA, 22-24 April 2008.