Quality of service control in soft real-time applications[1]

Luigi Palopoli, Tommaso Cucinotta
ReTiS Lab, Scuola Sup. Sant'Anna, Pisa, Italy

Antonio Bicchi
Facoltà di Ingegneria, Università di Pisa, Italy

Abstract

In this paper we present results obtained in the context of Quality of Service (QoS) control for soft real-time applications. The discussion addresses the issue of dynamically adjusting the bandwidth for a set of periodic tasks, when a reservation-based (RB) CPU scheduling policy is used. RB techniques are particularly suitable for this kind of applications since they allow an accurate mathematical modelling of the dynamic evolution of the QoS experienced by tasks. Based on this model, a control policy guaranteeing specified QoS levels for different tasks is illustrated, along with necessary and sufficient conditions for its existence. Moreover, the problem of steering a task QoS back into its nominal level is tackled, in response to deviations due to temporary overload conditions. Simulation results are reported, for the purpose of validating the approach.

1 Introduction

An emerging class of time-sensitive applications is implemented in computer systems by means of software tasks. Important examples are multimedia streaming programs, video/audio players, software sound mixers, etc.... Other examples are embedded systems used in data-intensive contexts, where relatively high volumes of sensor data are flowing and must be processed and analyzed in real time (i.e. radar systems). In such systems it frequently occurs that tasks share a pool of hardware/software resources, such as the CPU, RAM memory, disks, device drivers, data structures of the operating system and so forth. Therefore, a scheduling policy is needed to ensure compliance with real-time constraints of the tasks.

This work is focused on the management of a particular type of resource - the CPU - but most considerations are applicable to other types of resources. Traditional hard real-time CPU scheduling techniques are based on worst case assumptions for the computation demand of the tasks, since even a single failure in respecting real-time constraints is not deemed acceptable. However, for the applications considered in this work, hard real-time scheduling can be overly conservative because the computation demand of tasks exhibits large fluctuations in time. Moreover, occasional failures in respecting the timing constraints of a task can be acceptable, if the Quality of Service (QoS) provided by the application does not degrade beyond acceptable limits.

Thus, it is desirable to tune the scheduling policy based on average execution times. This way a larger number of tasks can be admitted to run onto a system, than a hard real-time policy would allow. Clearly, during the system lifetime, it is expected that occasionally the computation requirement by the tasks is higher than available, leading to deadline misses. Desirable features of soft real-time systems in such situations (*system overload*) are: 1) the scheduling anomalies due to the overload should be limited to the task that caused the overload (this property is called *temporal isolation*), 2) it should be possible to have at least a stochastic assessment of the behavior of the system (e.g. knowing the probability of deadline misses based on the stochastic distribution of the execution times). More generally, a desirable feature of soft real-time systems is to achieve a fluid allocation of the processor: each task should execute as if it were on a dedicated, slower processor regardless of the presence of other tasks. A good approximation of this abstraction is achieved by *CPU reservation* algorithms. They were first proposed in [10] for the CPU and have been implemented in a number of different systems using different scheduling algorithms [14, 1, 6, 15].

One of the biggest problems of CPU reservation systems is the correct allocation of CPU bandwidth to each task. In presence of wide variations of the required computation time, a static choice for the bandwidth might result, at different times, into a wasteful use of system resources or into degradations of the QoS. To cope with this problem, many authors proposed the use of feedback control mechanisms inside the operating system. A first proposal of this kind for time sharing systems dates back to 1962 [5]. More recently, feedback control techniques have been applied to real-time scheduling [11, 16, 7, 9] and multimedia systems [17, 2]. Owing to the difficulties in modeling schedulers as dynamic systems, these works could offer little analytical evidence of the effectiveness of their approaches.

This gap has been filled in, for the CPU reservations, in [3], where the authors showed that a CPU reservation is a discrete event system that can realistically be modeled as a switching and parametric dynamic system. Based on this result, in [12] a feedback control approach based on switching linear controllers was proposed. Both the controller's

[1]This work has been partially supported by the European OCERA IST-2001-35102 and RECSYS IST-2001-32515 projects.

synthesis and the closed loop system's performance were attacked using Lyapunov tools for quadratic stability, resulting into the definition of convex optimization problems. Potential drawbacks of this approach are: 1) convex optimization problems are heavy to solve on-line (within operating systems or middle-ware layers), 2) the system's analysis is based on merely sufficient conditions such as quadratic stability. Moreover, the application of linear controllers is not necessarily the best policy in terms of cost and performance.

In order to cope with these problems, in the present paper, we propose a different design approach. Specifications of QoS requirements are associated to regions in the state space of the system modeling the execution of each task. Control techniques are described to make such regions invariant and attractive. In this way, it is possible to provide the desired QoS whilst the bandwidth associated to each task remains close to its actual needs. An important problem, which is also addressed in the paper, is the presence of saturation constraints on the command variable due to the limited availability of CPU power. Despite its simplicity, the approach is applicable to a wide range of applications. Both the proposed analysis techniques and the control law are computationally inexpensive and have been implemented in a Linux based real-time system.

The paper is organized as follows. In Section 2 we will shortly review some basic concepts concerning real-time scheduling theory. In Section 3 we will formalize the QoS management as a control problem and in Section 4 we will propose our solutions. Finally some conclusions and discussions on future development are reported in Section 5

2 Background on real-time scheduling

Before developing a formal model for reservation-based scheduling, it is useful to introduce some definitions and to briefly discuss traditional scheduling solutions from which the former are derived.

2.1 The real-time task model

According to the real-time task model, a task $\mathcal{T}_i$ is a stream of jobs $J_i(k)$. Each job $J_i(k)$ arrives (becomes executable) at time $r_i(k)$, and finishes at time $f_i(k)$ after executing for a time $c_i(k)$. Moreover, $J_i(k)$ is characterized by a deadline $d_i(k)$, that is respected if $f_i(k) \leq d_i(k)$, and is missed if $f_i(k) > d_i(k)$. For the sake of simplicity, we will only consider *periodic tasks*, in which $r_i(k+1) = r_i(k) + T_i$, where T_i is the *task period*. Moreover, we will assume that $d_i(k) = r_i(k) + T_i$; hence, $r_i(k+1) = d_i(k)$. A quantity of interest for some real-time applications based on periodic tasks is the so called *jitter* on the finishing time defined as the difference: $f_i(k) - f_i(k-1) - T$. Having a small jitter allows one to consider the task as a fixed delay element and to use this information in the design of the overall system.

2.2 Reservation-based scheduling.

A reservation relative to a task $\mathcal{T}_i$ is represented by a pair (Q_i, T_i^s), meaning that, for each *reservation period* T_i^s following each job $J_i(k)$ arrival time $r_i(k)$, $\mathcal{T}_i$ is allocated a CPU time *budget* of Q_i, whenever in need. Defining the allocated CPU *bandwidth* as $B_i = Q_i/T_i^s$, the reservation can equally be represented by the pair (B_i, T_i^s).

Define the *virtual finishing time* $\mathrm{v}_i(k)$ as the time a job would finish if it were running on a dedicated processor of speed B_i times the real CPU speed. The time $J_i(k)$ becomes eligible for execution is given by $r_i(k)$ if $J_i(k-1)$ finished in due time and $v_i(k-1)$ otherwise. Hence,

$$v_i(k) = \begin{cases} r_i(k) + \frac{c_i(k)}{B_i} & \text{if } v_i(k-1) < r_i(k) \\ v_i(k-1) + \frac{c_i(k)}{B_i} & \text{otherwise.} \end{cases} \quad (1)$$

It is well known that a bandwidth reservation approximates a fluid allocation up to a granularity dictated by the choice of T_i^s. This is highlighted, for example, in [6], where it is proved that, for a job associated to a reservation (Q_i, T_i^S), it holds that:

$$v_i(k) - \delta \leq f_i(k) \leq v_i(k) + \delta, \quad (2)$$

with $\delta = T_i^s(1 - B_i)$. Throughout this paper, T_i^s will be assumed to be fixed while Q_i is a free parameter.

A reservation mechanism is often implemented on the top of a "host" hard real-time scheduler (e.g. Rate Monotonic or Earliest Deadline First [8]). This imposes constraints on the choice of the Q_i parameters for the task set. It has been proved that each task $\mathcal{T}_i$ attached to a reservation (B_i, T_i^s) is guaranteed to receive its reserved amount of execution time provided that: $\sum_{i=1,\ldots,n} B_i \leq U_l$, with $U_l \leq 1$ depending on the "host" scheduling algorithm.

2.3 Quality of service metrics

When dealing with soft real-time application it is imperative to formally define clear metrics to gauge the Quality of Service supplied by the tasks. This paper deals with two types of QoS requirements: 1) the scheduling error job $J_i(k)$ starts with, which is the deviation from the deadline experienced by $J_i(k-1) : f_i(k-1) - d_i(k-1)$, 2) the jitter $f_i(k) - f_i(k-1) - T$. It is useful to relate both quantities to the period considering the ratio $e_i(k) = (f_i(k-1) - d_i(k-1))/T$ and $F_i(k) = (f_i(k) - f_i(k-1) - T)/T$. The two quantities are related: $F_i(k) = e_i(k+1) - e_i(k)$. Furthermore, $e_i(k)$ is by definition lower bounded by -1.

The management of the QoS is more easily done considering the "virtual evolution". To this purpose introduce the *virtual scheduling error* $\epsilon_i(k)$ and the *virtual jitter* $\Phi_i(k)$ defined as:

$$\begin{aligned} \epsilon_i(k) &= \tfrac{v_i(k-1) - d_i(k-1)}{T} \\ \Phi_i(k) &= \tfrac{v_i(k) - v_i(k-1) - T}{T} = \epsilon_i(k+1) - \epsilon_i(k). \end{aligned} \quad (3)$$

Figure 1: Feedback control scheme for QoS management.

Considering the result in equation 2, it is possible to write:

$$\frac{T_i^s}{T_i} + \epsilon_i(k) \geq e_i(k) \geq \epsilon_i(k) - \frac{T_i^s}{T_i}$$
$$2\frac{T_i^s}{T_i} + \Phi_i(k) \geq F_i(k) \geq -2\frac{T_i^s}{T_i} + \Phi_i(k)$$

Thereby, assuming that the T_i^s/T_i ratio be small enough, it is possible to approximate the scheduling error and the jitter with their virtual counterparts. Hereinafter, we will no longer make this distinction. As a final remark, observe that bounding the evolution of the scheduling error $\epsilon_i(k)$ in a set $[-e, E]$ with $E \geq 0$ and $1 > e > 0$, amounts to bounding the jitter $\Phi_i(k)$ in the set $[-(e+E), (e+E)]$. Summing up, QoS requirements considered in this paper can be expressed as "acceptable" connected regions for the evolution of the virtual scheduling error $\epsilon_i(k)$.

3 Problem statement

Consider a set of periodic tasks $\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_n$ executing on the same processor. Job $J_i(k)$ has a computation time $c_i(k)$, which is randomly varying in a predefined set $[h_i, H_i]$. Suppose that after the termination of each job it is possible to decide the bandwidth B_i that the task will receive for the next job. The problem that we will cope with is how to take such a decision in order for a set of QoS requirements associated to the tasks to be respected.

The choice of a strategy is largely dependent on the characterization of the random process $c_i(k)$. With this respect, we consider two cases: the **deterministic bound** case, in which at each k it holds $Pr(c_i(k) \in [c_{mi}(k), c_{Mi}(k)]) = 1$, where $Pr(.)$ represents the probability and $[c_{mi}(k), c_{Mi}(k)]$ is a known interval; the **probabilistic bound** case, in which at each k it holds $Pr(c_i(k) \in [c_{mi}(k), c_{Mi}(k)]) = p < 1$ and $Pr(c_i(k) \notin [c_{mi}(k), c_{Mi}(k)]) = 1-p$, where $p/(1-p) \gg 1$. This classification covers a significant range of applications. Under these scenarios it is beneficial to dynamically adjust the bandwidth devoted to each job, upon its finishing time, by a QoS controller implementing a feedback control scheme. The QoS controller has a finite bandwidth availability, so the best strategy is one that allocates each job bandwidth based on all task states. This situation is depicted in Figure 1. A tough problem on this way is that QoS measures are collected asynchronously for the different tasks. So, at each point the controller can only rely on a partial information. Effective approaches to tackle this problem are currently under investigation.

In this paper, we take a simplified view and design a decentralized scheme where each task is endowed with a dedicated controller. The constraint $\sum B_i \leq U_l$ is simply enforced by requiring $B_i(k) \leq B_i^{(max)}$ with $\sum B_i^{(max)} \leq U_l$. As well as being a first step toward the construction of a more general theory, the simplification may have itself two practical applications, which in the general case make it preferable to the static allocation of a fixed bandwidth $B_i^{(max)}$. The first one is controlling the jitter within guaranteed bounds. The second one is keeping allocated bandwidth to the actual needs of the task at each time. The unused bandwidth thus reclaimed can be devoted to secondary tasks that execute *in background* with no guaranteed bandwidth. Before coming to a formal definition of goals and constraints for the controller, we show how to model a single CPU reservation as a dynamic system.

3.1 Dynamic Model of a Single Reservation.

Due to the properties of reservation based algorithms (i.e. the temporal isolation property), the dynamic model can be built for a single task regardless of other tasks present in the system. From now on, we adopt a simplified the notation by removing the task index from all the quantities. The evolution of the system is evaluated at the termination $f(k)$ of each job. The scheduling error $\epsilon(k)$ is considered as a state variable defined in the set $]-1, +\infty[$ while bandwidth $B(k)$ can be thought of as a command variable. The evolution of the scheduling error can be derived from Equation 1, and it is different depending on whether $J_i(k-1)$ terminates within the deadline or not. It can be easily proved that (see [3] for details) the scheduling error evolves according to:

$$\epsilon(k+1) = \begin{cases} \epsilon(k) + \frac{c(k)}{TB(k)} - 1 & \epsilon(k) \geq 0 \\ \frac{c(k)}{TB(k)} - 1 & \epsilon(k) < 0. \end{cases} \quad (4)$$

In the sequel, it will be useful to re-write the above as

$$\epsilon(k+1) = \begin{cases} \epsilon(k) + \frac{u(k)}{\tilde{u}(k)} - 1 & \epsilon(k) \geq 0 \\ \frac{u(k)}{\tilde{u}(k)} - 1 & \epsilon(k) < 0. \end{cases} \quad (5)$$

where we introduced $u(k) = 1/B(k)$ and the required bandwidth $\tilde{u}(k) = T/c(k)$. Thus, the control action $u(k)$ acts affinely on the state variable $\epsilon(k)$.

3.2 Constraints and goals

Recalling the discussion in the previous section, we will formalize QoS requirements by specifying an admissible connected region $\mathcal{E} = [-e, E]$, with $1 > e \geq 0$, and $E > 0$, for the evolution of $\epsilon(k)$. If one sets $e = 1$, then jitter requirements are irrelevant and the only requirement is on the maximum deviation from the deadline. The measure of $\mathcal{E}$ can also be regarded as a cost function: minimizing $\mid \mathcal{E} \mid$ corresponds to minimizing the jitter variations. Due to the occurrence of perturbing events it can be acceptable that the scheduling error can evolve for some time outside of $\mathcal{E}$. In this case, we require that the region be reachable under some control policy: i.e. there exists a sequence $u(k)$ that steers $\epsilon(k)$ back into $\mathcal{E}$.

Figure 2: Block diagram for QoS controller

As far as the command variable $u(k)$ is concerned, a saturation constraint is induced by the upper limitation $B^{(max)}$ introduced above. Therefore constraints on $u(k)$ can be generally expressed as: $u(k) \geq u^{(min)} \geq 1$.

Observe that an ideal controller, i.e. a controller knowing the computation time $c(k)$ of a job before executing it, would simply set $u(k) = \tilde{u}(k)$. Thereby, in order to ensure bandwidth reclaiming and to avoid excessive QoS fluctuations, it is desirable that the ratio $u(k)/\tilde{u}(k)$ varies by a "little" extent around 1. The latter property can be related to the existence of an invariant region $\mathcal{E}$ for $\epsilon(k)$ by the following fact, which is easy to show.

If $\epsilon(k) \in [-e, E]$, *then* $r_M \geq \frac{u(k)}{\tilde{u}(k)} \geq r_m$,

where $r_M = 1 + E$ and $r_m = max\{1 - (e + E), 0\}$.

Summarizing, constraints on the system evolution are: 1) $\epsilon(k) \in \mathcal{E}$, for some $\mathcal{E}$ chosen at the user's convenience; 2) $u(k) \geq u^{(min)}$. The design goals that we shall consider are: 1) constraining the evolution of $\epsilon(k)$ to a region $\mathcal{E}' \in \mathcal{E}$ of "minimum" measure; 2) constraining the evolution of the ratio between $u(k)/\tilde{u}(k)$ to a "small" interval $[r_1, r_2]$ with $r_1 \leq 1$ and $r_2 \geq 1$; 3) requiring that in response to perturbation $\epsilon(k)$ returns into $\mathcal{E}$ in minimum time.

4 Control Design

In this section, after introducing some definitions, the control design problem is tackled for both the deterministic and probabilistic bound cases. We denote by $\mathcal{U}$ the family of functions $u(c_m, c_M, \epsilon)$ of real arguments c_m, c_M, and ϵ, with $c_M > c_m$. Such functions represent feedback controllers that, at step k, read ϵ and decide the command variable u knowing that c will be in the range $[c_m, c_M]$. We assume that the bound $[c_m, c_M]$ is produced, step by step, by another system component, that is application dependent (see Figure 2). The subset of $\mathcal{U}$ respecting the constraint on the choice of u ($u(k) \geq u^{(min)}$) will be denoted as $\tilde{\mathcal{U}}$. The following definitions are adapted from standard set invariance theory [4]:

Definition 1 *Consider system 5 and let $\mathcal{A}$ and $\mathcal{B}$ be two connected subsets of the real set $\mathbb{R}$: $\mathcal{A}$ is said a* robustly controlled invariant *if, $\forall k$, there exists a feedback control law $u \in \tilde{\mathcal{U}}$ that, $\forall \epsilon(k) \in \mathcal{A}, \forall c(k) \in [c_m(k), c_M(k)]$, guarantees that $\epsilon(k+1) \in \mathcal{A}$; $\mathcal{A}$ is said* reachable *from $\mathcal{B}$ if there exists a feedback control law $u(c_m, c_M, \epsilon) \in \tilde{\mathcal{U}}$ s.t. $\forall \epsilon(k) \in \mathcal{B}$, $\exists n \in \mathcal{N}$ s.t. for all sequences $\{c(h) \in [c_m(h), c_M(h)]\}$ with $h = k, k+1, .., k+n-1$, it holds $\epsilon(k+n) \in \mathcal{A}$; $\mathcal{A}$ is said* globally reachable *if it is reachable from $]-1, +\infty[$.*

4.1 Deterministic bound

In this subsection we find conditions for existence of a controlled invariant and globally reachable set in case of deterministic bounds for variations of $c(k)$.

Theorem 1 *Let $\mathcal{E} = [-e, E]$ be an interval of the real set with $1 \geq e > 0$ and $E > 0$, let α be defined as: $\alpha = \inf_k \left\{ \frac{c_m(k)}{c_M(k)} \right\}$, and $c_M = \sup_k \{c_M(k)\}$. $\mathcal{E}$ is a controlled invariant set for System 5 if and only if:*

$$e + \alpha E \geq 1 - \alpha \wedge \frac{T}{c_M} \geq u^{(min)} \tag{6}$$

Proof: The proof is constructive in that it shows the family of controllers making the set $\mathcal{E}$ invariant. First, we consider invariance of $\mathcal{E}$ with respect to a single step k. This can be seen as a result of a game between two players: the controller, that chooses $u(k)$, and the disturbance, that chooses $c(k)$. The controller plays first knowing $\epsilon(k)$ and the range of possible moves $[c_m(k), c_M(k)]$ for the disturbance. For each possible move c of the disturbance and for each $\epsilon(k) \in \mathcal{E}$ there is a set of legal moves (i.e. respecting $u(k) \geq u^{(min)}$) for the controller, denoted by $\mathcal{U}(c, \epsilon(k))$, ensuring that $\epsilon(k+1)$ will be in $\mathcal{E}$. As the controller plays first, the latter condition can be guaranteed if and only if it chooses a move in the set $\bigcap_{c \in [c_m(k), c_M(k)]} \mathcal{U}(c, \epsilon(k))$. Hence, $\epsilon(k+1)$ remains in $\mathcal{E}$ if and only if $\forall \epsilon(k) \in \mathcal{E}$ the intersection $\bigcap_{c \in [c_m(k), c_M(k)]} \mathcal{U}(c, \epsilon(k))$ is not empty.

For the sake of brevity, in the following we consider only the case $e + E < 1$. Let $\alpha(k)$ be defined as $\alpha(k) = \frac{c_m(k)}{c_M(k)}$. The computation of the set $\bigcap_{c \in [c_m(k), c_M(k)]} \mathcal{U}(c, \epsilon(k))$ is done considering that the range of admissible $u(k)$ ensuring $E \geq \epsilon(k+1) \geq -e$ is given by: $\frac{T}{c_M(k)}(1 + E - s(\epsilon(k))) \geq u(k) \geq \frac{T}{c_M(k)}(1 - e - s(\epsilon(k)))$, where $s(x) = x$ if $x > 0$ and $s(x) = 0$ if $x \leq 0$. The intersection over c of the set of acceptable command variables $\{u(k)\}$ is given by:

$$\left\{ u \in \mathbb{R} \; s.t. \begin{cases} \frac{T}{c_M(k)}(1 + E - s(\epsilon(k))) \geq u \\ u \geq \frac{T}{c_m(k)}(1 - e - s(\epsilon(k))) \\ u \geq u^{(min)} \end{cases} \right\} \tag{7}$$

Such an intersection is not empty if and only if:

$$e + \alpha(k) E \geq 1 - \alpha(k) \wedge \frac{T}{c_M(k)} \geq u^{(min)}.$$

The proof is ended considering the intersection of the constraints for all possible k.

■

Figure 3: First row: computation times c(k). Second row: scheduling error evolution. Third row: allocated CPU Bandwidth B(k)

Corollary 1 *The smallest possible controlled invariant set is given by:* $E = 0$ *and* $e = (1 - \alpha)$.

Remark 1 *If the evolution of* $\epsilon(k)$ *is constrained to the smallest controlled invariant and* $c(k)$ *varies in a small set, then the ratio* $\frac{u(k)}{\tilde{u}(k)}$ *is also constrained in the small set* $[\alpha, 1]$*, thus complying with the second design goal stated in section 3.2.*

The following results show how to steer the system from an arbitrary initial state into a controlled invariant set. Their proof is omitted for the sake of brevity, and can be found in [13].

Theorem 2 *Let* $\mathcal{E} = [-e, E]$ *be a controlled invariant set for System 5, and let the average maximum execution time* $\tilde{c}_M$ *be defined as:* $\tilde{c}_M = lim_{N->+\infty} \frac{1}{N} \sum_1^N c_M(k)$. *A sufficient condition for the global reachability of* $\mathcal{E}$ *is* $T > u_m \tilde{c}_M$*, while a necessary condition is* $T \geq u_m \tilde{c}_M$. *Moreover, if the condition* $T > u_m c_M$ *holds, with* $c_M = sup_k\{c_M(k)\}$*, then there exists a control law that steers the scheduling error from an initial value* $\epsilon(0) > E$ *into* $\mathcal{E}$ *in at most* $n = \left\lceil \frac{\epsilon(0) - E}{1 - u^{(min)} c_M / T} \right\rceil + 1$ *steps.*

Remark 2 *In the case in which the range* $[c_m(k), c_M(k)]$ *is constant and equal to* $[h, H]$*, it is possible to prove that* $\mathcal{E}$ *is globally reachable for System 5 if and only if* $T > u^{(min)} H$.

4.1.1 Numerical example: The proposed techniques were validated by both simulation and implementation in the Linux kernel. For demonstrative purposes we report here some simulation results. The assumed variations for the computation time were relative to a random walk: $c(k-1) + \Delta > c(k) > c(k-1) - \Delta$, i.e. $[c_m, c_M] = [c(k-1) - \Delta, c(k+1) + \Delta]$. The $u(k)$ control variable has been chosen in the middle of the set of possible values identified in the proof of theorem 1.

Figure 3 reports the job computation times along with lower and upper bounds (h and H) in the first row. The second row reports the evolution of the scheduling error with respect to the controlled invariant set limits: the error starts from outside of the invariant region, due to a temporary system overload, then it is steered back into it. The third row shows the allocated bandwidth during the system evolution.

4.2 Probabilistic bound

The control strategy for the deterministic bound is based on the definition of a controlled invariant set $\mathcal{E}$ which can be reached by using an appropriate control law. The basic result is that the tighter is the bound $[c_m(k), c_M(k)]$ the smaller can $\mathcal{E}$ be made. This simple strategy can be adapted for tasks respecting the probabilistic bound model. Indeed, using this model, the process $c(k)$ evolves in the fixed interval $[h, H]$, but it resides most of the time in a predictable tighter interval that we will assume as constant for the sake of simplicity: $\forall k, [c_m(k), c_M(k)] = [c_m, c_M]$. Therefore, it is possible to devise a control algorithm using two regions: $\mathcal{E}_I = [-e_I, E_I]$ and $\mathcal{E} = [-e, E]$, with $\mathcal{E}_I \subseteq \mathcal{E}$. The control algorithm has to satisfy the following rules: 1) it has to make $\mathcal{E}$ invariant, i.e. $\forall \epsilon(k) \in \mathcal{E}$ and $\forall c(k) \in [h, H]$, $\epsilon(k+1) \in \mathcal{E}$ must hold; 2) it has to make $\mathcal{E}_I$ invariant whenever $c(k)$ respects the tighter bound, i.e. $\forall \epsilon(k) \in \mathcal{E}_I$ and $\forall c(k) \in [c_m, c_M]$, $\epsilon(k+1) \in \mathcal{E}_I$ must hold; 3) it has to make $\mathcal{E}_I$ attractive from $\mathcal{E}$ if $c(k)$ respects the tighter bound for a sufficient number of steps, i.e. $\forall \epsilon(k) \in \mathcal{E} \backslash \mathcal{E}_I$ a finite natural n must exist s.t. for all sequences $\{c(h) \in [c_m, c_M]\}$ with $h = k, k+1, .., k+n-1$, $\epsilon(k+n) \in \mathcal{E}_I$ holds. The existence of a control law respecting these rules is stated in the following result, which is a straightforward application of theorems 1 and 2.

Corollary 2 *Let* $\mathcal{E}_I = [-e_I, E_I]$ *and* $\mathcal{E} = [-e, E]$ *be two intervals of the real set with* $1 \geq e_I > 0$, $1 \geq e > 0$, $E > 0$, $E_I > 0$, *and* $\mathcal{E}_I \subseteq \mathcal{E}$. *Then there exists a feedback control algorithm respecting the above cited three rules iff:*

$$\begin{cases} e + \alpha E \geq 1 - \alpha & (a) \\ e_I + \alpha_I E_I \geq 1 - \alpha_I & (b) \\ e + \beta_1 E_I \geq 1 - \beta_1 & (c) \\ e_I + \beta_2 E \geq 1 - \beta_2 & (d) \\ \frac{T}{H} > u^{(min)} & (e) \end{cases}, \begin{cases} \alpha = \frac{h}{H} \\ \alpha_I = \frac{c_m}{c_M} \\ \beta_1 = \frac{h}{c_M} \\ \beta_2 = \frac{c_m}{H} \end{cases} \quad (8)$$

The conditions found in the previous theorem can be used to set up an optimization problem for which different cost functions can be used. A possible cost function can be a weighted sum of the measures of the two regions. Another interesting possibility to evaluate is the ratio between $\mathcal{E}_I$ and $\mathcal{E}$. This topic is currently under investigation.

5 Conclusions and future work

In this work we showed the application of control theoretical approaches to the management of the QoS supplied by a set of task sharing a common CPU. Tasks are scheduled using the resource reservation algorithms; in this framework each task can be modeled as an uncertain switching discrete-time system, whose evolution is observed upon the occurrence of discrete event systems (i.e. the termination of each job). We proposed the application of a feedback controller to each task, that adjusts the bandwidth allocated to each job based on the Quality of Service experienced by

the previous job and on the prediction of a variability range for the computation demand of the job. The proposed techniques are very simple, and thus implementable with low computational cost inside operating systems or middle-ware layers.

This work is only the first step toward the construction of a more complete theory. There are at least two open research issues that will be investigated in the near future. The first one is a more complete study of the probabilistic bound model: in particular it is of paramount importance to evaluate the steady state probability for the system to evolve in the region dictated by the QoS specification. Another challenging goal is to develop control strategies taking into account the global evolution of all tasks.

References

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

[2] Luca Abeni and Giorgio Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *Proceedings of the IEEE Real Time Computing Systems and Applications*, Hong Kong, December 1999.

[3] Luca Abeni, Luigi Palopoli, Giuseppe Lipari, and Jonathan Walpole. Analysis of a reservation-based feedback scheduler. In *Proc. of the Real-Time Systems Symposium*, Austin, Texas, November 2002.

[4] F. Blanchini. Set invariance in control. *Automatica*, 1999.

[5] F. J. Corbato, M. Merwin-Dagget, and R. C. Daley. An experimental time-sharing system. In *Proceedings of the AFIPS Joint Computer Conference*, May 1962.

[6] G.Lipari and S.K. Baruah. Greedy reclaimation of unused bandwidth in constant bandwidth servers. In *IEEE Proceedings of the 12th Euromicro Conference on Real-Time Systems*, Stokholm, Sweden, June 2000.

[7] B. Li and K. Nahrstedt. A control theoretical model for quality of service adaptations. In *Proceedings of Sixth International Workshop on Quality of Service*, 1998.

[8] C. L. Liu and J. Layland. Scheduling alghorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), 1973.

[9] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Marley. Performance specifications and metrics for adaptive real-time systems. In *Proceedings of the 21th IEEE Real-Time Systems Symposium*, Orlando, FL, December 2000.

[10] Clifford W. Mercer, Stefan Savage, and Hideyuki Tokuda. Processor capacity reserves for multimedia operating systems. Technical Report CMU-CS-93-157, Carnegie Mellon University, Pittsburg, May 1993.

[11] Tatsuo Nakajima. Resource reservation for adaptive qos mapping in real-time mach. In *Sixth International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, April 1998.

[12] L. Palopoli, L. Abeni, and G. Lipari. On the application of hybrid control to cpu reservations. In *Hybrid systems Computation and Control (HSCC03)*, Prague, april 2003.

[13] Luigi Palopoli and Tommaso Cucinotta. QoS control in reservation-based scheduling. Technical Report ReTiS-TR-03-02, Scuola Superiore S. Anna, 2003.

[14] Raj Rajkumar, Kanaka Juvva, Anastasio Molano, and Shuichi Oikawa. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.

[15] Dickson Reed and Robin Fairbairns (eds.). Nemesis, the kernel – overview, May 1997.

[16] John Regehr and John A. Stankovic. Augmented CPU Reservations: Towards predictable execution on general-purpose operating systems. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS 2001)*, Taipei, Taiwan, May 2001.

[17] David Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proceedings of the Third usenix-osdi*. pub-usenix, feb 1999.