Adaptive reservations in a Linux environment *

T. Cucinotta, L. Palopoli, L. Marzario, G. Lipari
ReTiS Lab, Scuola Sup. Sant'Anna, Pisa, Italy

L. Abeni
Pisa, Italy

Abstract

In this paper, we address the problem of adaptively reserving the CPU to concurrent soft real-time tasks, in order to meet target Quality of Service requirements. First, we present two new techniques inspired to the idea of stochastic control. Then, we present a flexible and modular software architecture suitable for adaptive scheduling, realised as a minimally invasive set of modifications to the Linux Kernel. Finally, we show experimental results that validate our approach and prove its effectiveness in the context of multimedia applications.

1. Introduction

Software based implementation of a class of time-sensitive applications is gaining momentum because it is generally regarded as cheaper and more flexible than a dedicated hardware solution. Important examples can be found in the area of consumer electronics: multimedia streaming programs, video/audio players, software sound mixers, movie editing, and so on. Such applications are characterised by implicit timing constraints for which occasional timing failures can be tolerable, provided that they do not become too frequent. Whenever applications of this kind populate the same system and compete for a pool of shared resources, an appropriate real-time scheduling solution is needed to attain both an efficient use of the processor and an acceptable level of Quality of Service (QoS) for the different tasks. To this regard, traditional real-time techniques, such as Rate Monotonic (RM) and Earliest Deadline First (EDF) [12], are not appropriate because they deem unacceptable even a single deadline violation, and schedulability tests are based on worst case assumptions. As a consequence, adoption of such techniques in the context of soft real-time applications leads to an overly conservative management of the CPU. On the other hand, it is crucial that large fluctuations on the execution or inter arrival times of a task do not affect the performance levels granted to other tasks.

In the past years, several scheduling algorithms have been proposed that mimic a fluid allocation of the processor, giving each task the "illusion" of running on a dedicated slower CPU. These range from Proportional Share (PS) [10] to Reservation Based (RB) [15] algorithms.

This work is based on the use of RB techniques for CPU reservation, which have been implemented on a variety of systems using different algorithmic solutions [19, 3, 11, 14]. We are confident that most of the presented results can be extended to the management of other kind of resources, like network bandwidth and disk, and to algorithms providing temporal protection other than CPU reservations, such as the ones based on PS.

The traditional way for using RB scheduling is to reserve a fixed fraction of the CPU utilisation to each task, so that its temporal constraints can be fulfilled. However, a static allocation is not effective if the task widely changes its execution requirements throughout its execution. In fact, an allocation based on "average" requirements would result into transient but unacceptable degradations of the provided QoS, while an allocation based on worst case assumptions would most times be inefficient in terms of CPU utilisation. This problem can be addressed by dynamically adapting the amount of resources reserved to each task, i.e. by using a feedback inside the scheduling mechanism.

A first proposal for feedback based scheduling of time sharing systems dates back to 1962 [9]. More recently, feedback control techniques have been recently applied to real-time scheduling [20, 13, 8, 7] and multimedia systems [23]. Owing to the difficulties in modelling schedulers as dynamic systems, these works only provide a limited mathematical analysis of the closed-loop performance, often based on approximate models or intuitive arguments. The application of feedback to RB algorithms was pioneered in [4] introducing the concept of *adaptive reservations*. This work opened up a new research thread. In [6], it is shown how it is possible to write an exact mathematical model for the dynamic evolution of a single reservation and to design a switching Proportional Integer (PI) controller based on a linearisation of the system. Stability results and synthesis techniques for tuning the parameters of the switch-

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

ing PI controller, based on the theory of hybrid systems and on convex optimisation, were shown in [16]. The problem was further investigated in [18], where a nonlinear feedback control scheme taking advantage of the specific structure of the system model was shown.

In this paper, we first introduce two novel control techniques, which have been designed by attacking the problem in the stochastic domain.Then, we describe a software architecture suitable for our feedback control strategies and we present its implementation in the Linux kernel. Contrary to other approaches that are more focused on hard real-time applications, such as RTlinux [22] and RTAI [21], we can run time-sensitive applications in user space with obvious benefits in terms of safety and access to a wealth of libraries available for Linux. Modifications of the original kernel have been carried on in such a way that ordinary Linux applications can run without any awareness whatsoever of the new environment.

2. Problem presentation

2.1. The tasks model

We consider a set of independent tasks $\mathcal{T}^{(1)}, ..., \mathcal{T}^{(n)}$ sharing a CPU. A task $\mathcal{T}^{(i)}$ consists of a stream of jobs, or instances, $J_k^{(i)}{}_{k\in\mathfrak{N}}$. Each job $J_k^{(i)}$ arrives (becomes executable) at time $r_k^{(i)}$, and finishes at time $f_k^{(i)}$ after executing for a time $c_k^{(i)}$. Job $J_k^{(i)}$ is associated a deadline $d_k^{(i)}$, which is respected if $f_k^{(i)} \leq d_k^{(i)}$, and is missed if $f_k^{(i)} > d_k^{(i)}$.

For our purposes, the sequences of computation times $\{c_k^{(i)}\}_{k\in\mathbb{N}}$ are considered as discrete-time continuous valued stochastic processes. For the sake of simplicity, we will restrict to *periodic tasks*, in which $r_{k+1}^{(i)} = r_k^{(i)} + T^{(i)}$, where $T^{(i)}$ is the *task period*. Moreover, we will assume that $d_{k+1}^{(i)} = d_k^{(i)} + T^{(i)}$; hence, $r_{k+1}^{(i)} = d_k^{(i)}$.

2.2. Resource Based Scheduling

In the application that we will show in this paper tasks are scheduled by a Reservation Based (RB) policy [15].

In a RB framework, a task $\mathcal{T}^{(i)}$ is associated a pair $(Q^{(i)}, P^{(i)})$, said *reservation*, meaning that the scheduling algorithm guarantees to $\mathcal{T}^{(i)}$ a *budget* of $Q^{(i)}$ execution time units in every *reservation period* $P^{(i)}$ (whenever in need). The ratio $b^{(i)} = Q^{(i)}/P^{(i)}$ is referred to as *bandwidth*. Dealing with periodic tasks, it is convenient to choose $P^{(i)}$ so that $T^{(i)} = kP^{(i)}, k \in \{1, 2, \ldots\}$. If the task is not allowed to execute for more than $Q^{(i)}$ units every $P^{(i)}$, even in presence of an idle processor, then the reservation is said *hard* [19]. In this paper we will restrict our attention to this class of RB algorithms, even though most techniques and considerations shown in the sequel are applicable to a good extent also to other types of reservations.

A very important property ensured by RB scheduling is the so called *temporal isolation*, i.e. a task's schedulability depends only on the behaviour of the task itself and on the assigned budget $Q^{(i)}$. Thanks to this property, the task can be thought of as running on a *virtual CPU* having speed a fraction $b^{(i)}$ of the CPU speed. In fact, defining the *virtual finishing time* $\mathrm{v}_k^{(i)}$ as the time the k^{th} job would finish if it were running on a virtual CPU with speed $b^{(i)}$, the enforcement of a hard reservation policy implies the following relation [11]:

$$v_k^{(i)} - \delta \leq f_k^{(i)} \leq v_k^{(i)} + \delta, \tag{1}$$

where $\delta = (1 - b^{(i)})P^{(i)}$. The above shows that in principle a RB scheduler can be made to approximate a "fluid" allocation of the processor as closely as needed by choosing $P^{(i)}$ small enough. However, in practical implementations, the overhead of context switches becomes relevant if $P^{(i)}$ is too small. A consistency relation necessary for a RB scheduler to work properly is

$$\sum_i b^{(i)} \leq U_{lub}, \tag{2}$$

with $U_{lub} \leq 1$ depending on the algorithm used for the implementation.

2.3. Adaptive Reservations

When considering soft real-time applications it is of paramount importance to quantify the Quality of Service that each task experiences during his execution. In our model we can tolerate occasional deadline misses as long as the anomaly is kept in check. Therefore, it is reasonable to define a quality of service metric, that we will call *scheduling error*, related to the deviation of the finishing time from the deadline. A possible definition for such a metric could be $e_k^{(i)} = (f_{k-1}^{(i)} - d_{k-1}^{(i)})/T_i$, where $e_k^{(i)}$ is the scheduling error experienced by job $J_{k-1}^{(i)}$. An ideal bandwidth allocation would be one for which $e_k^{(i)} = 0$ for all k. Indeed, both $e_k^{(i)} > 0$ and $e_k^{(i)} < 0$ are undesirable situations, since in the former the task does not respect its timing constraint, whilst in the latter it receives an excess of bandwidth that would better be allocated to other activities.

When dealing with tasks which expose a large fluctuation of the computation requirement, a static allocation of bandwidth to the task is not appropriate. An adaptation mechanism is needed to dynamically allocate the bandwidth to a task during its execution, thus the idea of *adaptive reservation*. In particular, in the line of research initiated in [4], we perform bandwidth adaptation using conceptual tools borrowed from feedback control theory. This concept is henceforth referred to as *feedback scheduling*.

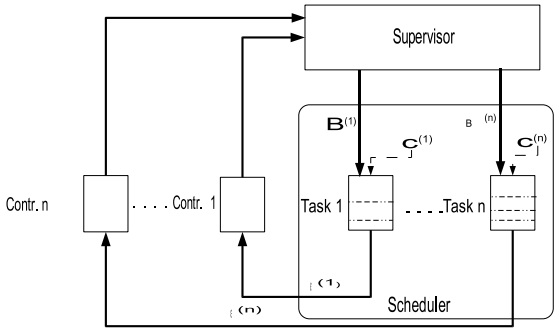

Figure 1. Envisioned architecture.

2.4. Dynamic model

In order to design a feedback control we need a mathematical model for the system dynamic evolution. To this regard, the scheduling error as defined above, although an appealing QoS metric, turns out to be cumbersome to use. Instead, we shall define a different metric, by approximating the actual finishing time f_k of each job with its *virtual* finishing time, $v_k : \varepsilon_k^{(i)} = \frac{v_{k-1}^{(i)} - d_{k-1}^{(i)}}{T^{(i)}}$. In view of Equation (1), it is easy to show that $\varepsilon_k^{(i)}$ constitutes an approximation of the original metric $e_k^{(i)}$:

$$\varepsilon_k^{(i)} - \delta^{'} \leq e_k^{(i)} \leq \varepsilon_k^{(i)} + \delta^{'}, \tag{3}$$

(where $\delta^{'} = \frac{\delta}{T} = (1 - b^{(i)})\frac{P^{(i)}}{T^{(i)}}$), which clearly shows that the introduced approximation is acceptable provided that the ratio $\frac{P^{(i)}}{T^{(i)}}$ be small enough. The dynamics of $\varepsilon_k^{(i)}$ is given by [6]:

$$\varepsilon_{k+1}^{(i)} = S(\varepsilon_k^{(i)}) + \frac{c_k^{(i)}}{T^{(i)} b_k^{(i)}} - 1 \tag{4}$$

where $S(x) = 0$ if $x < 0$ and $S(x) = x$ if $x \geq 0$.

For most RB algorithms, $\varepsilon_k^{(i)}$ is exactly and easily measurable upon the termination of each job.

2.5. Control goal

For what said above, the control goal is to keep the scheduling error evolution as close to zero as possible. Modelling the scheduling error evolution as a stochastic process, reasonable design goals for the QoS can be formulated on:

- the first order probability density distribution $f_{\varepsilon_k^{(i)}}(\cdot)$: it can be used to visually compare performance of two different control algorithms;
- the expected value of the s.e. $\mu_{\varepsilon_k^{(i)}}$ and its variance $\sigma^2_{\varepsilon_k^{(i)}}$: these values can be used for a quantitative comparison of two control techniques;
- the probability for the scheduling error $\varepsilon_k^{(i)}$ to fall in a specified segment $[-e^{(i)}, E^{(i)}]$ of the real axe.

3. Feedback scheduling techniques

Equation (4) describes a first order switching system, in which $\varepsilon_k^{(i)}$ is a measurable state variable that we want to control, the bandwidth $b_k^{(i)}$ acts as a command variable, whereas $c_k^{(i)}$ is an exogenous disturbance term. As a matter of fact, we have a collection of first order systems that evolve asynchronously one another, their states being observed at asynchronous points in time (jobs termination for the different tasks).

The asynchronism of the system makes it difficult to design a global controller. A simpler choice is a decentralised scheme where a dedicated controller decides the bandwidth of each task looking at the evolution of the task itself in isolation. This idea is not completely applicable since the bandwidths chosen by the different controllers undergo a global constraint dictated by Equation (2). A minor departure from the entirely decentralised scheme is to include a supervisor that, whenever the controllers violate the constraint, resets the values of the bandwidths to fix the problem (e.g. operating a weighted compression or a saturation). From the standpoint of each controller, every time the supervisor is forced to act an impulsive disturbance is experienced (see Figure 1).

3.1. Single controller general design

The control scheme just introduced consists of a collection of controllers attached to each task and a supervisor that performs corrective actions only when a controller chooses a value for the bandwidth in contrast with Equation (2) determining an overload condition. The latter component is described in depth in [2] and we will omit further details. Rather, this section is mainly concerned with the design of the dedicated controllers. In order to reduce the probability of overload conditions, and the subsequent supervisory corrections, each controller is constrained by a "local" saturation constraint: $b_k^{(i)} \leq B_{max}^{(i)}$.

From now on, we will concentrate on how to design a controller for a single task and the (i) superscript will be dropped for notational convenience. We propose a scheme based on two components (see Figure 2): a *predictor* that, upon the termination of J_{k-1}, supplies a set of parameters I_k related to a prediction of c_k; a *controller* that decides the bandwidth b_k based on the set of parameters I_k and on the

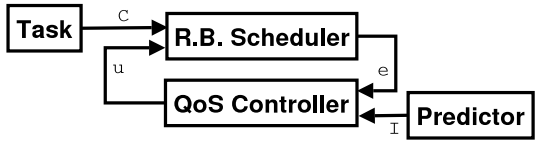

Figure 2. Block diagram for QoS controller

measurements of ε_k collected from the scheduler. The predictor plays in this scheme an important role: the more accurate the prediction the better the resulting control performance. The ability to build an accurate predictor is related to the stochastic properties of the input process. A very simple predictor is one which is based on statistics (e.g. moving average) gathered on the past computation times. Actually, we will show that the type of information that the predictor needs to supply depends on the control scheme.

In the rest of the section we first quickly review the invariant based control, which has already been presented in [18, 17], then we introduce two new approaches, namely the stochastic dead beat and cost optimal controllers, which are based on the stochastic formulation of the problem.

3.2. Invariant based design

The goal of an invariant based controller is to constrain the scheduling error evolution within a small region $[-e, E]$, compensating for the fluctuations of c_k. The information I_k provided at each step by the predictor is a range $[h_k, H_k]$ where the next computation time c_k is expected to fall. The invariant based controller guarantees that, as long as $c_k \in [h_k, H_k]$, the scheduling error remains constrained in $[-e, E]$. Otherwise, a *recovery* mode is used to steer back the error into the invariant region. A theoretical discussion on conditions for such a controller to exist as well as on the problem of mistaken predictions (i.e. $c_k \notin [h_k, H_k]$) can be found in the cited paper. In this context we just summarise results on how to choose the bandwidth:

step k) choose b_k

$$\begin{cases} \in \left[\frac{H_k}{T(1+E-S(\varepsilon_k))}, \frac{h_k}{T(1-e-S(\varepsilon_k))}\right], \text{if } \varepsilon_k \leq \varepsilon^1 \\ \in \left[\frac{H_k}{T(1+E-S(\varepsilon_k))}, B_{max}\right], \text{if } \varepsilon^1 < \varepsilon_k \leq \varepsilon^2 \\ = B_{max}, \text{if } \varepsilon_k > \varepsilon^2 \end{cases} \tag{5}$$

where $\varepsilon^1 = 1 - e - \frac{h_k}{TB_{max}}$ and $\varepsilon^2 = 1 + E - \frac{H_k}{TB_{max}}$.

step 0) choose b_0 in the same range as for a negative scheduling error.

The control formula just showed embeds the simplest recovery policy, which assigns the maximum available bandwidth in such situations. Alternative policies are discussed further in [17].

3.3. Stochastic dead beat approach

This control scheme attacks the design problem in the stochastic domain. The goal is to choose a bandwidth such that the expectation of the next scheduling error be equal to a desired value. The expectation that we are considering is conditioned to the past evolution of the system. If the desired value is zero we refer to the controller as Stochastic Dead Beat (SDB). It is possible to prove that the control law having such a property, and satisfying the saturation constraint, can be expressed as follows:

$$b_k = \begin{cases} \frac{\mu_{C_k}}{T(1-s(\varepsilon_k))} & \text{if } \varepsilon_k < 1 - \frac{\mu_{C_k}}{TB_{max}} \\ B_{max} & \text{if } \varepsilon_k \geq 1 - \frac{\mu_{C_k}}{TB_{max}} \end{cases}. \tag{6}$$

If $\varepsilon_k > 1 - \frac{\mu_{C_k}}{TB_{max}}$, then it is not possible to guarantee that the expected next error be zero. The information I_k required from the predictor is μ_{C_k}, i.e. the expectation of c_k conditioned to the past evolution of the system. This can be done, for example, with a moving average performed on last execution times. Despite its simplicity, this technique is able to achieve a very good performance, as we will show in Section 5.

3.4. Optimal cost (OC) approach

This technique takes inspiration from dynamic programming techniques, in that the controller optimises, at each step, the expectation (conditioned to the past evolution of the system) of a cost function $w(\varepsilon, b)$. Such cost is associated to a transition of the system using $b_k = b$ that results in $\varepsilon_{k+1} = \varepsilon$. In particular we chose a cost function accounting for the deviation of the next scheduling error from zero, and the bandwidth being used: $w(\varepsilon_{k+1}, b) = \gamma\varepsilon_{k+1}^2 + (1-\gamma)b$, where $\gamma \in]0, 1[$ allows to assign different weights to the two cost components. The following formula [17] gives the optimum bandwidth choice in the general case:

$$b_k(\varepsilon_k) = \sqrt[3]{\rho + \delta(\varepsilon_k)} + \sqrt[3]{\rho - \delta(\varepsilon_k)}, \quad \rho = \frac{\gamma(\sigma^2 + \mu^2)}{(1-\gamma)}$$

$$\delta(\varepsilon) = \sqrt{\left(\frac{\gamma}{1-\gamma}\right)^2 (\sigma^2 + \mu^2)^2 + \left(\frac{2}{3}\frac{\mu\gamma[1 - S(\varepsilon_k)]}{1-\gamma}\right)^3}$$

μ and σ denote, respectively, the mean value and standard deviation of c_k (conditioned to the past evolution of the system), which must be provided by the predictor component. This formula can be directly used for all $\varepsilon_k \leq \varepsilon^* = 1 + \frac{3}{2\mu_C}\sqrt[3]{\frac{\gamma}{1-\gamma}(\sigma^2 + \mu^2)}$, which is the range for which $\delta(\varepsilon_k)$ is real. For $\varepsilon_k > \varepsilon^*$, the formula still holds if computations are properly performed in the complex domain. Furthermore, note that the optimum bandwidth value found with this formula is subject to the usual saturation constraint.

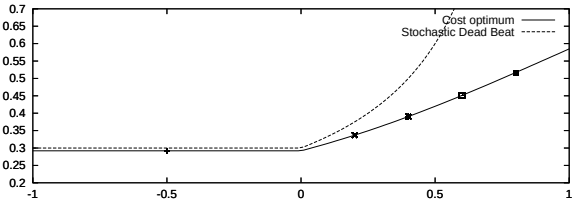

Figure 3. $B(\varepsilon)$ **laws for the OC and SDB cases.**

Figure 3 reports the optimal $B(\varepsilon_k)$ function for a particular set of parameters. The same figure makes also a comparison with the bandwidth function in (6).

An important problem with this approach is that the computation of the bandwidth requires several floating point operations for which it is not immediate to achieve an efficient kernel implementation. For fixed μ and σ the problem is relatively simpler in that it is possible to do efficient linear interpolations of the curve. For dynamically changing parameters, more sophisticated techniques are required and they are currently under investigation.

4. Architecture of OCERA implementation

The applications to be scheduled with the techniques considered so far are soft real-time by nature. Thus it is of paramount importance to provide an implementation of such techniques in the context of a general purpose OS, where such applications are mostly used. We proved this to be feasible by providing a reference implementation in the Linux kernel, which is described in this section.

The implementation has been carried out in the context of the OCERA project (see http://www.ocera.org), financially supported by the European Commission under the IST programme, fifth framework. The aim of the OCERA project is to enhance the real-time characteristics of Linux for both hard and soft real-time systems by providing a set of open software components. In this paper we will focus on the soft real-time components, describing how adaptive reservations are implemented.

In OCERA, soft real-time tasks are implemented as regular Linux processes running in user space, and loadable modules are used to implement resource reservations as shown in Figure 4. The `QRES` module implements resource reservation according to the CBS algorithm [1]. The `QSPV` module implements the admission control policy, and it provides the supervisor functionality for the single QoS controllers, which are implemented in turn as separate modules. This orthogonal separation among the different components allows a great level of configurability.

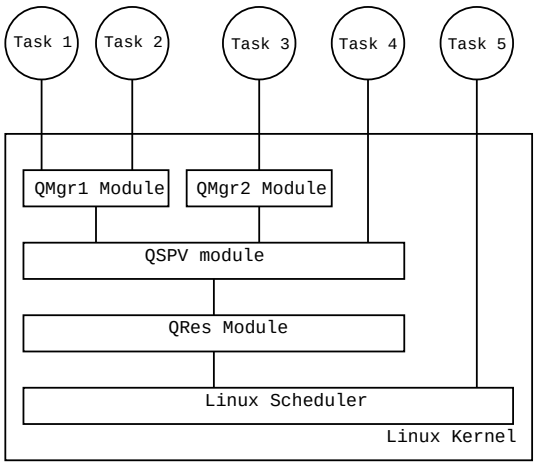

Figure 4. OCERA soft RT architecture.

4.1. Generic Scheduler Patch (GSP)

Scheduling functionality of the Linux kernel has been extended through the application of a non-intrusive patch, namely the Generic Scheduler Patch. This defines an interface for allowing other modules to intercept scheduling related events and undertake appropriate actions in response to them. For this purpose, the GSP defines a set of `hooks`, i.e. function pointers, one for every scheduling event, made available to other modules by exporting the necessary symbols.

The scheduling events for which hooks are defined are job arrivals, job finishings, process creations, and process terminations, for which the defined hooks are, respectively: `unblock`, `block`, `fork` and `cleanup`. The `setsched` hook is invoked when the scheduling policy is changed by calling the `sched_setscheduler()` system call. Initially, all hooks are unset in the kernel, and are not used.

Thus, modules that need to customize the scheduler behaviour, like our QRES module, may set the hooks to point to their appropriate handlers implementing the new behaviour. A more detailed description of the implementation of the generic scheduler patch and of the scheduling module can be found in [5].

4.2. Scheduling Module

The `QRES` module implements a slightly modified version of the CBS algorithm [3], where the original soft reservation paradigm has been replaced with a hard reservation one (see [19] for a comparison between the two approaches), which allows to use the system evolution model introduced in Section (2.4). The CBS belongs to the class of RB algorithms described in Section (2.2), and is based on the earliest deadline first algorithm (EDF) [12]. Due to

the underlying EDF strategy, and the overhead introduced into the kernel by our modules, the U_{lub} parameter in Equation (2) can be set greater than 0.95 for most practical experiments.

Once the QRES module has been loaded into the kernel, a task can require to be scheduled according to a RB policy by invoking the `sched_setscheduler()` system call with the policy SCHED_CBS. After such a call, the QRES hook handlers will intercept all scheduling events related to that task, implementing the desired scheduling policy. The `sched_setscheduler()` system call is also used by the task to specify scheduling parameters, through the use of an extended version of the structure `sched_param`. Specifically, the task is required to provide the desired budget and period into this structure.

4.3. QoS Supervisor (QSPV) Module

When a new reservation is created specifying a certain budget Q and a certain period P, the system must check if there is enough free bandwidth to accommodate for the new reservation. This admission control is performed by the QoS supervisor module. This module intercepts all the calls to the `scheduler_setsched()` via the `setsched` hook.

Three different flavours of this module exist, each one implementing a different admission control policy: *saturation*, *compression* and *reject*. They differ in their response to requests that cannot be accommodated. In all cases, if the sum of the CPU utilisations of the existing reservations, plus the utilisation of the new reservation, does not exceed U_{lub}, then the request is forwarded to the `setsched` handler of the QRES module; the `sched_setscheduler()` succeeds and the task is scheduled according to the CBS algorithm with the specified parameters.

If there is not enough bandwidth to serve the new request, the action depends on the selected policy:

saturation policy: the highest possible budget is assigned to the task so that the total CPU utilisation does not exceed U_{lub}.

compression policy: all the reservations are recomputed ("compressed") so that we can make enough space for the new request. See [4] for a detailed description of the compression algorithm.

reject policy: the `sched_setscheduler()` returns with an error and the task is scheduled in background.

The QSPV module is also used by the QoS Manager to dynamically change the budget of an existing reservation according to the feedback control algorithm.

4.4. The QoS Manager module

We provide different QoS management modules (denoted with QMGR1 and QMGR2 in Figure 4) that can coexist in the same system. Each module provides a different controller strategy and can serve more than one task.

A task can choose the QoS manager for its execution by specifying, in the `sched_setscheduler()` call, the SCHED_QMGR1 or SCHED_QMGR2 scheduling policy, and by providing proper parameters to the module through the `sched_param` structure.

In order to use adaptive reservations, a task must be structured in the following way. At the beginning, the task must set the scheduling policy by calling the `sched_setscheduler` system call, specifying for example the QMGR1 policy, and providing scheduling parameters. The `setsched_hook` of the QMGR1 module is then invoked. After storing the required parameters in its internal data structures, the QMGR1 module invokes (see Figure 5) the `qspv_request_create()` function of the QSPV module to initialise the reservation budget and period.

After initialisation, the task enters a loop. Each execution of the loop corresponds to a *job* of the task. For example, in case of a MPEG decoder, a job may correspond to the decoding of one frame. At the end of the loop, the task invokes the `qmgr_end_cycle()` function. Such a call results in the activation of the `setsched` hook into the kernel, which results in a call to the proper handler into the QMGR1 module. This obtains the amount of budget consumed by the job by calling the `qres_get_consumed()` function provided by the QRES module. Then, the control law is applied and a new budget is computed and set with the `qspv_change_budget()` function of the QSPV module. If there is not enough free bandwidth to accommodate for the new budget, the QoS supervisor applies one of the three possible behaviours described in the previous section, with the difference that if the reject policy is in place then the old value is kept unchanged. Finally, the task blocks waiting for the next periodic event by calling the `wait_period()` function.

5. Experimental results

In this section we report experimental results gathered on a real Linux system. We modified the *xine* (see http://xine.sourceforge.net) MPEG player, in order to control, for each frame, the finishing time of the decoding stage.

In the first experiment, we show the benefit of adopting a feedback scheduling mechanism as opposed to a static allocation of the bandwidth. Then, we compare the performance of the various controllers.

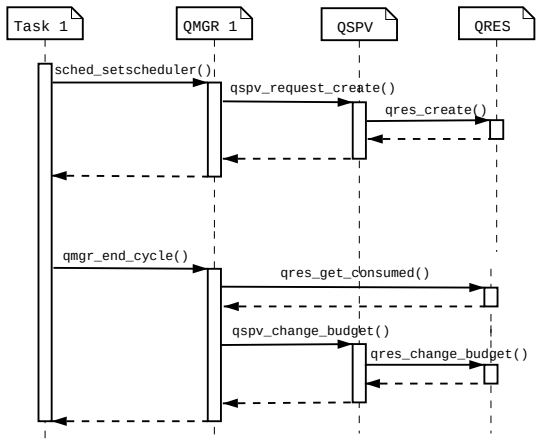

Figure 5. Interaction between the QMGR, the QSPV and the QRES modules.

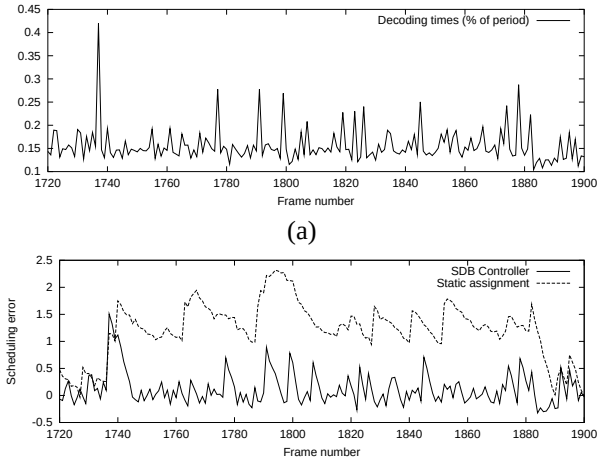

Figure 6. Computation times gathered from the Xine player (a), and scheduling error resulting with a static and dynamic bandwidth assignments (b).

Consider the MPEG decoding times shown in Figure 6 (a), which evolve around the 15% of the task period. The scheduling error evolution achieved with a static bandwidth assignment equal to 16% is shown in Figure 6 (b), compared to the one achieved with a SDB controller, where the expected value μ_{C_k} is approximated by the predictor by performing a moving average of the last ten samples, and the saturation value has been set to 17%. A visual comparison between the two evolutions is illustrative of the extent of the achieved enhancement.

In this section we compare the performance of different controllers: 1) the switching PI controller [6, 13], 2) the invariant based controller [18], 3) the stochastic dead-beat controller, 4) the optimal cost controller. The experimental Probability Density Functions (PDF) resulting from the application of the four controllers are reported in Figure 7 and Figure 8. A preliminary work was to *hand-tune* the parameters of the controllers so as to optimise their performance (in order to get a fair comparison).

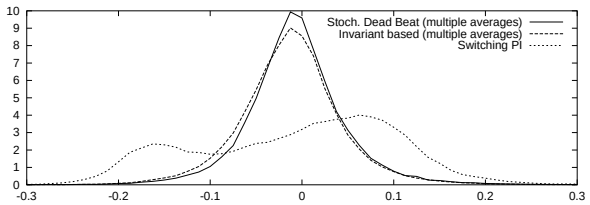

Figure 7. Scheduling error PDF achieved by switching PI, invariant-based and SDB controllers.

As Figure 7 demonstrates, the switching PI controller is largely outperformed by both the invariant based and the stochastic dead beat schemes. This is also partially due to the use, inside the last two approaches, of a predictor component taking advantage of the periodic structure of the decoding times, by using multiple moving averages. The comparison between the SDB and the invariant based solution reveals very similar performance. A strength of the SDB is its extreme simplicity, while the invariant based solution allows for more flexibility to the price of a greater design complexity.

In Figure 8 we compare the SDB and optimal cost approaches, this time using a simpler predictor for all schemes, which just computes a single moving mean over last ten samples. A visual comparison between the performance of the SDB in Figure 7 and Figure 8 gives an idea of the impact of the predictor quality on the overall controller performance. Regarding the optimal cost controller, we report the results for two values of the γ parameter. In the first case we chose $\gamma = 0.90$, thus weighing very much the importance of the scheduling error. With respect to the SDB case, the PDF is shifted to the right; this is because the optimal cost controller achieves a trade-off between bandwidth consumption and performance. Attaching even more importance to the bandwidth ($\gamma = 0.75$), the curve is further shifted to the right. Performance in terms of the scheduling error degrades, but the system tends to "save" bandwidth for other tasks. As pointed out earlier, this flexibility is paid in terms of computational complexity.

6. Conclusions and future work

In this paper, we addressed the problem of controlling the QoS in soft soft real-time applications by using reser-

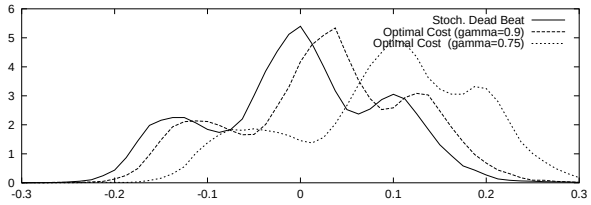

Figure 8. Scheduling error PDF achieved by different stochastic based control schemes.

vation based scheduling techniques augmented by feedback control strategies. First, we proposed two new control strategies that are inspired to the ideas of stochastic control. We compared the performance of these new controllers with previous approaches. Second, we described a software architecture for feedback control in the Linux OS, which has been realised as a minimally set of invasive changes to the Linux scheduler, plus a set of external modules. We showed results obtained applying the proposed techniques to an MPEG player, demonstrating their effectiveness.

In the short future, we want to consider alternative architectural solutions, such as the possibility of moving part of the QoS manager functionality from kernel to user space. Furthermore, we plan to attack the problem of scheduling tasks that use multiple resources (disk, network, etc.) in a coordinate way, by applying RB techniques for a coordinated schedule of them.

References

[1] L. Abeni. Server mechanisms for multimedia applications. Technical Report RETIS TR98-01, Scuola Superiore S. Anna, 1998.

[2] L. Abeni. *Supporting time-sensitive Activities in a Desktop Environment*. PhD thesis, Scuola Superiore S. Anna, December 2002.

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

[4] L. Abeni and G. Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *Proc. of the IEEE Real Time Computing Systems and Applications*, Hong Kong, December 1999.

[5] L. Abeni and G. Lipari. Implementing resource reservations in linux. In *Proc. of Fourth Real-Time Linux Workshop*, Boston, MA, December 2002.

[6] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole. Analysis of a reservation-based feedback scheduler. In *Proc. of the Real-Time Systems Symposium*, Austin, Texas, November 2002.

[7] G. T. C. Lu, J. Stankovic and S. Son. Feedback control real-time scheduling: Framework, modeling and algorithms. *Special issue of RT Systems Journal on Control-Theoretic Approaches to Real-Time Computing*, 23(1/2), September 2002.

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

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

[10] P. Goyal, X. Guo, and H. M. Vin. A hierarchical cpu scheduler for multimedia operating systems. In *Proc. of the 2nd OSDI Symposium*, October 1996.

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

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

[13] 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 *Proc. of the 21th IEEE Real-Time Systems Symposium*, Orlando, FL, December 2000.

[14] L. Marzario, G. Lipari, P. Balbastre, and A. Crespo. IRIS: A new reclaiming algorithm for server-based real-time systems. To appear in Proc. of RTAS 2004, May 2004.

[15] C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves for multimedia operating systems. Technical Report CMU-CS-93-157, Carnegie Mellon University, Pittsburg, May 1993.

[16] 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.

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

[18] L. Palopoli, T. Cucinotta, and A. Bicchi. Quality of service control in soft real-time applications. In *Proc. of the IEEE 2003 conference on decision and control (CDC02)*, Maui, Hawai, USA, December 2003.

[19] 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.

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

[21] http://www.aero.polimi.it/ rtai/. RTAI Official Website.

[22] http://www.fsmlabs.com. FSMLabs - RTLinux Official website.

[23] D. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proc. of the Third usenix-osdi*. pubusenix, feb 1999.