A robust mechanism for adaptive scheduling of multimedia applications

TOMMASO CUCINOTTA*, LUCA ABENI†, LUIGI PALOPOLI†, GIUSEPPE LIPARI*
*Scuola Superiore Sant'Anna, †University of Trento

We propose an adaptive scheduling technique to schedule highly dynamic multimedia tasks on a CPU. We use a combination of two techniques: the first one is a feedback mechanism to track the resource requirements of the tasks based on "local" observations. The second one is a mechanism that operates with a "global" visibility, reclaiming unused bandwidth. The combination proves very effective: resource reclaiming increases the robustness of the feedback, while the identification of the correct bandwidth made by the feedback increases the effectiveness of the reclamation. We offer both theoretical results and an extensive experimental validation of the approach.

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: Special-purpose and application-based systems—*Real-time and embedded systems*; D.4.7 [**Software**]: Operating systems - Organization and design—*real-time systems and embedded systems*; D.4.8 [**Software**]: Operating systems - Performance—*Stochastic analysis*; J.7 [**Computer applications**]: Computers in other systems—*real-time*

General Terms: Design, Performance, Experimentation

1. INTRODUCTION

In recent years, personal computers have made inroad in the domain of multimedia applications. The increasing computational power and flexibility of modern PCs enable an effective sharing of hardware resources between concurrent applications with obvious cost reductions. An interesting example is offered by video encoding: using a computer endowed with a modern operating system, it is possible to encode multiple streams at once, using a different concurrent task for each stream, operating with different qualities and different coding standards.

When the multimedia information has to be processed in real-time (e.g., for a video-conferencing system), the ability for each task to meet its timing constraints has a very important impact on the Quality of Service (QoS) perceived by the user.

Authors' addresses: T. Cucinotta and G. Lipari, ReTiS Lab, Scuola Superiore Sant'Anna, Piazza dei Martiri della Libertà, Pisa, Italy; L. Abeni and L. Palopoli, DISI, University of Trento, Via di Sommarive 14, Povo (TN), Italy
This work has been supported by the European Commission by means of the FRESCOR (`http://www.frescor.org` FP6/2005/IST/5-034026, IRMOS (`http://www.irmosproject.eu`) FP7/2008/ICT/214777 and CHAT FP7/2008/ICT/224428 European Projects.
This project has also been supported by the Provincia Autonoma di Trento by means of the RoSE PAT/CRS Project.

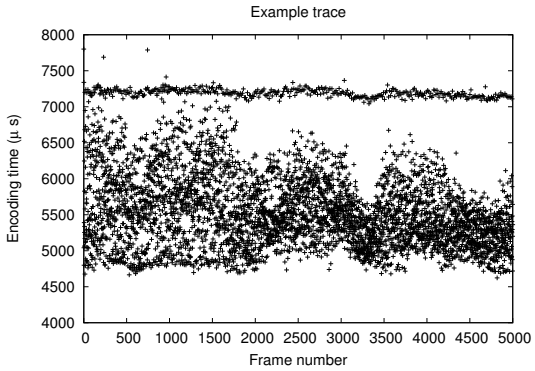

Fig. 1. Trace of execution times from the execution of a MPEG encoder.

Contrary to the safety critical applications considered by the classical theory of real-time systems [Liu and Layland 1973], occasional violations of timing constraints do not invalidate the correctness of the computation as long as the anomaly is kept in check. Designers are then confronted with a problem of challenging complexity: how to schedule the CPU making an efficient use of its computation power and providing the tasks with a controlled level of timing performance.

It is commonly argued that an effective tool to tackle this problem is a scheduling algorithm enabling one to reserve a given fraction of the CPU to each task (from here on referred to as *bandwidth*). However, bandwidth reservation is not *per se* a conclusive solution. Indeed, the bandwidth reserved to each task has to be sufficient to accommodate its computing requests, which are very much dependent on the data the task processes and can be highly time–varying. As an example, consider the trace of execution times for an MPEG encoding application reported in Figure 1. As well as the "natural" variations due to the different type of frames in the Group of Pictures (GOP), we can clearly see "structural variations" due to changes in the scene. For instance, the average computation time clearly decreases from frame 1500 to frame 2500. This trend continues in the frames from 2500 to 5000. In a situation like this, 1) it is difficult to make an initial choice for the bandwidth, 2) even when this information is available, a static choice for the bandwidth could be, at some point in time, wasteful and, at some others, insufficient.

A possible approach to deal with this problem is the so-called *feedback scheduling*: a feedback controller operates on top of the scheduler adapting the scheduling parameters according to the sensed evolution of the task's timing behaviour. A mechanism like this can be made more effective using a combination of sensing (on the current state of the task) and of predictions (on the future evolution of the workload). This idea underpins the approach known as "adaptive reservations", which is one of the cornerstones of the construction presented in this paper.

Adaptive reservations (AR), as presented in previous work [Abeni and Buttazzo 1999; Abeni et al. 2004], suffer two major limitations. The first one is on the performance guarantees that can be provided: since an AR has a limited scope (it only looks at the evolution of one task), it can offer performance guarantees only when its bandwidth requests are totally granted. When the system is heavily loaded, as a result of the interference with other adaptive reservations, such guarantees can in principle be disrupted. The second limitation is that, since AR use predictions,

the performance can be weakened by prediction errors.

A second approach is endowing the scheduler with a reclaiming mechanism [Lipari and Baruah 2000; Lin and Brandt 2005]. When a task uses less bandwidth than the one it is allocated, the extra bandwidth can be redistributed to the other tasks. Contrary to AR, this mechanism leverages a visibility of the state of the system as a whole. However, its application is not sufficient for two reasons. First, the mechanism can only operate when the system is not overloaded (otherwise there is no bandwidth to reclaim). Second, if the bandwidth allocated to the tasks is not close to their actual needs, the mechanism is not able to compensate. This is particularly true for periodic tasks with different periods, as discussed in Section 6.

1.1 Paper contributions

The most important contribution of this paper is to show a beneficial synergy between two different mechanisms: adaptive reservations and a global supervisor that implements a reclaiming policy. On one hand the robustness of adaptive reservations can be improved by providing additional bandwidth if available; on the other hand the reclaiming mechanism can be more effective by a correct allocation of bandwidth to the tasks. In order for these goals to be fulfilled, some important conditions have to be met on the design of the two components. As far as the design of adaptive reservations is concerned, the control law has to guarantee a level of performance to each task regardless of the presence of other tasks, provided that "a contract" with the global supervisor is respected. We propose a theoretical analysis whereby such guarantees can indeed be offered as far as the contract is respected.

The supervisor is required to properly manage overload conditions, in which the bandwidth requests of the different tasks have to be re-scaled without ever violating the contract. When the available bandwidth exceeds the request of the adaptive reservation control loop, the reclaiming mechanism of the supervisor has to re-distribute it to the tasks with a small latency and "fairly", so that the increased robustness deriving from the extra bandwidth is received by all tasks. Another important contribution of this paper is a resource reclaiming algorithm, called SHRUB, that operates with a policy of this kind.

1.2 Paper structure

The paper is organised as follows. In Section 2, we offer background information on the tasking model, on the scheduling algorithm that we use and on how it is used in the adaptive reservations approach. In Section 3 we provide an overview of our approach. In Section 4, we describe the particular type of AR proposed in this paper, putting the stress on its performance guarantees. In Section 5, we describe our global supervisor showing how it can manage overload conditions and how it reclaims bandwidth using the SHRUB algorithm. In Section 6, we highlight the problem arising from the use of reclaiming alone through a simple example, then in Section 7, we report results from our experimental validation conducted over a real implementation of the proposed mechanism. Finally, In Section 8, we describe the most relevant related work, and in Section 9 we offer our conclusions.

2. BACKGROUND INFORMATION

In this paper, we consider a set of real-time tasks $\{\tau_i\}$ running on a shared CPU. Even though the discussion that follows refers to a *single CPU*, the approach presented in this paper is equally applicable to the case of multi-processor systems, scheduled by a *partitioned* real-time scheduling policy where real-time tasks are statically bound to each CPU. An example of this kind is the partitioned EDF scheduler proposed by Faggioli et al. [Faggioli et al. 2009].

A real-time task τ_i is activated multiple times, generating a sequence of jobs $J_{i,j}$. Each job $J_{i,j}$ arrives (becomes executable) 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 $f_{i,j} > d_{i,j}$. In this paper, we focus on multimedia applications, for which deadlines are considered *soft* constraints, i.e., a few violations are deemed acceptable and lead to a performance degradation, rather than cause severe faults (like in hard real-time systems). In this case, reasonable performance metrics can be related to the frequency (or the probability) of a deadline miss or to the maximum deviation of the finishing-time from the deadline.

We focus on *periodic* tasks, where arrival times are spaced out by a *task period* T_i, i.e., $r_{i,j+1} = r_{i,j} + T_i$, and each activation time is also the deadline of the previous instance $d_{i,j} = r_{i,j} + T_i = r_{i,j+1}$. For example, a video-conferencing application where each frame is acquired from a camera at a fixed rate, encoded, transmitted, decoded and displayed, fits well in this model.

As multiple real-time tasks may be concurrently active at the same time, a real-time scheduling mechanism is used to properly schedule the CPU among them. To this purpose, we advocate the use of *resource reservations*. Each task τ_i is associated a reservation (Q_i, P_i), with the meaning that τ_i is allowed to execute for Q_i time units (*budget*) in every interval of length P_i (*reservation period*). The bandwidth allocated to the task is $B_i = Q_i/P_i$. It is important not to confuse the reservation period P_i with the task period T_i : although $P_i = T_i$ is a perfectly reasonable assignment, it is often useful to set the reservation period so that $T_i = N_i P_i, N_i \in \mathbb{N}$.

In this paper, we use a novel reservation-based scheduler called SHRUB, that may be considered as an evolution of the Constant Bandwidth Server (CBS) [Abeni and Buttazzo 1998]. In CBS, reservations are realised by means of an Earliest Deadline First (EDF) scheduler which schedules tasks $\{\tau_i\}$ based on their *scheduling deadlines* $\{d_i^s\}$, dynamically managed by the CBS algorithm. When a new job $J_{i,j}$ arrives, the server checks whether it can be scheduled using the last assigned deadline, otherwise the request is assigned an initial deadline equal to $r_{i,j} + P_i$. Each time the job executes for Q_i time units (i.e., its budget is depleted), its scheduling deadline is postponed by P_i. Thereby, each task is prevented from executing for more than Q_i units with the same deadline, and it is guaranteed a minimum bandwidth of $B_i = Q_i/P_i$ regardless of the behaviour of the other tasks, under the assumption that the following *schedulability condition* holds:

$$\sum_i B_i \leq U_{max}, \tag{1}$$

where $U_{max} \leq 1$ is a user defined constant. For scheduling algorithms based on EDF (as CBS and SHRUB), in theory U_{max} can be set equal to 1. However, from

Symbol	Meaning	Symbol	Meaning
τ_i	i-th task	$Q_{i,j}$	reservation budget for job $J_{i,j}$
$J_{i,j}$	j-th job of task τ_i	$B_{i,j} = \frac{Q_{i,j}}{P_i}$	reserved bandwidth for job $J_{i,j}$
$c_{i,j}$	computation time of job $J_{i,j}$	$\epsilon_{i,j}$	scheduling error of job $J_{i,j}$
$\mathcal{C}_i = \{\mathcal{C}_{i,j}\}$	stochastic process for $\{c_{i,j}\}$	$\mathcal{E}_i = \{\mathcal{E}_{i,j}\}$	stochastic process for $\{\epsilon_{i,j}\}$
T_i	activation period of task τ_i	$H_{i,j}$	percentile estimate for $\mathcal{C}_{i,j}$
P_i	reservation period for τ_i	$\overline{B}_i$	min. bandwidth guaranteed to τ_i
N_i	integer number s.t. $T_i = N_i P_i$	$\overline{Q}_i$	min. budget guaranteed to τ_i

Table I. Main symbols used in the paper (subscript i is omitted in the paper when the discussion refers to a specific task).

a practical perspective, one may want to limit the maximum amount of bandwidth reserved to real-time tasks choosing a value for U_{max} smaller than 1. For example, this may be useful to avoid starvation of the underlying OS activities in case of overload of the real-time tasks. Further details about SHRUB follow in Section 5.2.

For the reader's convenience, Table I summarises the most important symbols used throughout the paper.

2.1 Dynamic model for the scheduling error

A CPU reservation can be considered as a discrete-event dynamic system whose evolution is observed at the termination of each job $J_{i,j}$. In this work, we propose to use the scheduling budget Q_i as a control input: while the reservation period P_i is held constant, the budget $Q_{i,j}$ can be freely assigned for each job $J_{i,j}$ to meet the QoS goals. This choice, as discussed next, allows us to build a clear dynamic model describing the evolution of the QoS as a function of the control input $Q_{i,j}$ and of an exogenous disturbance term given by the computation time $c_{i,j}$. The parameter P_i can be used to decide the granularity of the CPU allocation (a smaller value for P_i corresponds to a more fluid allocation but to a greater overhead).

Instrumental to this construction is the definition of the *scheduling error* as the difference between the server scheduling deadline $d^s_{i,j}$ (evaluated at the finishing-time of each job) and the soft deadline of the task:

$$\epsilon_{i,j} \triangleq d^s_{i,j} - d_{i,j} = d^s_{i,j} - r_{i,j} - T_i. \tag{2}$$

A positive value for $\epsilon_{i,j}$ means that $J_{i,j}$ finished late, receiving less bandwidth than it needed. Conversely, a negative value means that it finished earlier than its deadline, so the assigned bandwidth was greater than needed. A null value corresponds to a perfect match between the task workload and the resource assignment.

As shown in previous work [Abeni et al. 2002], in a hard CBS an approximation for the dynamic evolution of the scheduling error is given by:

$$\epsilon_{i,j+1} = S(\epsilon_{i,j}) + \left\lceil \frac{c_{i,j+1}}{Q_{i,j+1}} \right\rceil P_i - T_i \tag{3}$$

where the function $S(\cdot)$ is defined as: $S(x) \triangleq \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$.

Figure 2 shows an example of scheduling error evolution: a task with a constant computation time of 240 time units for the first three jobs ($c_{1,1} = c_{1,2} = c_{1,3} = 240$) and a period of $T_1 = 100$ time units (equal to its relative deadline) is served by

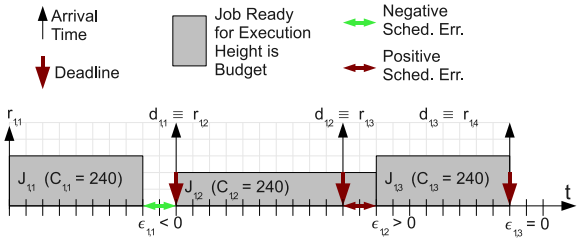

Fig. 2. Example of scheduling error evolution.

a reservation having a constant period of $P_1 = 10$ time units (corresponding to the tics on the `x` axis), and a variable budget equal to $Q_{1,1} = 3$ time units for the first job, $Q_{1,2} = 2$ for the second one and $Q_{1,3} = 3$ again for the third one (corresponding to the height of the greyed boxes in the drawing). The corresponding scheduling error values of $\epsilon_{1,1} = -20$, $\epsilon_{1,2} = 30$ and $\epsilon_{1,3} = 0$, computed by means of Equation (3), are also highlighted in the drawing (horizontal thick little arrows). The presence of the $S(\cdot)$ function in the dynamic model for the scheduling error evolution models the fact that, if a job completes late (e.g., $J_{1,2}$ in the example), then its tardiness accumulates on the response-time of the next job. However, if it completes early (e.g., $J_{1,1}$ in the example), then the next job is not affected, because its activation does not happen before the begin of the subsequent period. Therefore, negative scheduling error values are immediately "forgotten" by the system, from a scheduling error evolution perspective. Recording a negative value for $\epsilon_{i,j}$ can be useful, from the perspective design perspective, because it can help reconstruct the time series of the computation times used in our control scheme.

Thanks to the fact that we keep the reservation period constant, we can simplify the notation by normalising the scheduling error to the server period. Therefore, $\epsilon_{i,j}$ will actually be a shorthand for $\frac{\epsilon_{i,j}}{P_i}$ and the model will be described by:

$$\epsilon_{i,j+1} = S(\epsilon_{i,j}) + \left\lceil \frac{c_{i,j+1}}{Q_{i,j+1}} \right\rceil - N_i. \tag{4}$$

Note that, with the boundary condition $\epsilon_{i,0} = 0$ (the first job completion time $\epsilon_{i,1}$ cannot be delayed by any previous job), in the above evolution model $\epsilon_{i,j}$ is constrained to be an integer variable: $\epsilon_{i,j} \in \mathbb{N}$.

The model described above assumes a constant bandwidth throughout the job execution. However, it can easily be generalised [Cucinotta et al. 2008] to the case in which changes are allowed, without ever violating Condition (1).

3. APPROACH OVERVIEW

The solution we advocate to cope with time-varying and unknown workload from the task is sketched in Figure 3. It consists of an adaptive scheduling mechanism based on the combination of a set of task controllers that formulate bandwidth requests for the each task, and of a global mechanism (*resource supervisor*) that manages the CPU using a global visibility of the different requests from the tasks.

Local controllers use a combination of a *feedback controller* and of a *predictor*. Each feedback controller monitors the so called *scheduling error*, which quantifies the deviation of each job from its timing constraints, and is responsible for controlling the evolution of such quantity within proper bounds. The predictor provides

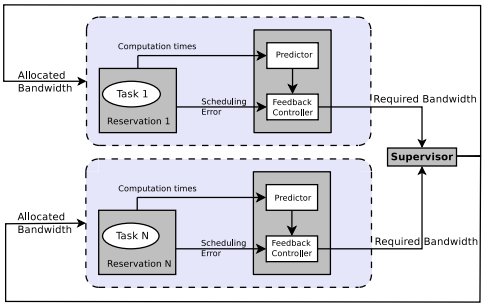

Fig. 3. The main architectural blocks of the proposed solution

an estimate of the computation time expected for the next job.

The resource supervisor ensures a correct and efficient utilisation of the scheduler. Namely, if the bandwidth requests coming from the different tasks exceed the total availability of CPU, a *compression* function re-scales the value of the bandwidth granted to the tasks within the limits. On the contrary, if at some point in time the total required bandwidth is below the total availability, an *expansion* function is used to maximise the CPU utilisation. The additional bandwidth received by each task increases the robustness in meeting its timing constraints (e.g., alleviating the adverse impact of an incorrect prediction).

Our proposed design for the task controller and the resource supervisor, along with the performance guarantees that can be offered to each task making specific assumptions on their interaction, will be discussed in the following sections.

4. CONTROL SCHEME AND ANALYSIS

Given the system in Equation (4), we aim at a control law providing guarantees on the evolution of the scheduling error (and ultimately on the delays introduced on each job). A hard real-time approach, based on the worst case execution time of the tasks, would lead to an over-provisioning of resources, which for multimedia applications is neither acceptable nor actually needed.

In contrast, for soft real-time systems, one can take two different approaches. The first one is providing *probabilistic guarantees*: the deadline of each job is met with a given probability. The second one is *deterministic*: a task can occasionally fail to meet its deadline but the maximum delay is bounded and the duration of this anomaly is limited.

As discussed next, we make a joint use of both techniques. Note that, for simplifying notation, the subscript i will be henceforth removed whenever the discussion is referred to a specific task τ_i.

4.1 Formal control goals

The probabilistic guarantees on the scheduling error evolution may be formalised as follows. Let us consider the sequence of computation times $\{c_j\}$ as a particular realisation of a *stochastic process* $\{\mathcal{C}_j\}$, and the sequence of scheduling errors $\{\epsilon_j\}$ resulting from Equation (4) as a realisation of the stochastic process $\{\mathcal{E}_j\}$. Then, whenever computing the control action at step j, we may state the following formal control goal for the controller:

$$\mathbf{Pr}\{\mathcal{E}_{j+1} \leq 0\} \geq \pi \tag{5}$$

where π is the minimum desired probability of keeping the scheduling error evolution within a *stability region* $\{\epsilon_j \leq 0\}$, i.e., meeting the deadline constraint.

However, such requirement is not easy to fulfil, because of the difficulty in computing in closed form the probability in the left-hand. Therefore, first we focus on a weaker requirement, that, as it will be shown later, is easier to fulfil:

$$\mathbf{Pr}\left\{\mathcal{E}_{j+1} \leq 0 \mid \mathcal{E}_j = \epsilon_j\right\} \geq \pi. \tag{6}$$

In other words, we require that the probability of meeting the deadline in the next step *given* the current value of the scheduling error be lower bounded. Since the controller has to compensate for the accumulated delay using additional bandwidth, the equation above can only be enforced if the scheduling error is inside a region (*attractivity basin*) $\{\epsilon_j \leq R\}$, with $R \in \mathbb{N} \cup \{0\}$. The shortcoming of such a requirement is that it specifies the behaviour of the system only when $\epsilon_j \leq R$.

For this reason, the requirement has been complemented with a deterministic requirement on the trajectories followed by ϵ_j over a time horizon, which can be formalised as follows. Assume that, at a time j_0, the scheduling error exits the attractivity basin, i.e., $\epsilon_{j_0-1} \leq R \wedge \epsilon_{j_0} > R$. Then, we want: 1) to steer it back, in a maximum number of steps L, into a region such that the requirement (6) can be fulfilled; 2) the maximum deviation of the scheduling error to be bounded by a value ϵ^{MAX}. These two requirements can be formally expressed as:

$$\begin{cases} \exists \tilde{j} \in j_0 + 1, \ldots, j_0 + L \text{ s.t. } \mathbf{Pr}\left\{\mathcal{E}_{\tilde{j}+1} \leq 0 \mid \mathcal{E}_{\tilde{j}} = \epsilon_{\tilde{j}}\right\} \geq \pi \\ \forall j \in j_0, \ldots, \tilde{j}, \epsilon_j \leq \epsilon^{MAX}. \end{cases} \tag{7}$$

As discussed at the end of the section, the attainment of both requirements in Equations (6) and (7), whenever possible, allows us to compute a bound for the unconditioned probability in Equation (5).

4.2 Control law

We now focus on how a task controller can be designed that formulates a correct bandwidth requirement to attain the goals above looking at one task *in isolation*.

The control law implemented by the task controller relies on a bound H_{j+1} of the computation time for the next job. Such value is provided by a *predictor* component. The predictor can use information on the past history of the computation times (recorded by the scheduler) and use the standard general purpose techniques developed for time-series prediction [Falk et al. 2006]. A different possibility for the predictor is to use application specific techniques (such as the one developed by Roitzsch and others [Roitzsch and Pohlack 2006] for MPEG decoding times) that execute a very quick preliminary analysis on the data item to be processed to infer the computation time required for processing it. Our particular solution is discussed in Section 7.

At each step j, our control law assigns the budget (request) Q_{j+1} for the next job based on the current value of the scheduling error ϵ_j and on the H_{j+1} estimate as follows:

$$Q_{j+1} = Q(\epsilon_j, H_{j+1}) = \begin{cases} \frac{H_{j+1}}{N - S(\epsilon_j)} & \text{if } \epsilon_j \leq E_j \\ PU_{max} & \text{otherwise.} \end{cases} \tag{8}$$

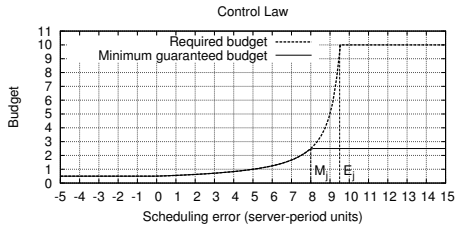

Fig. 4. Example of required and granted budget values, at varying ϵ_j values, for $H_{j+1} = 5, P = 10, N = 10$.

where $E_j \triangleq N - \left\lceil \frac{H_{j+1}}{PU_{max}} \right\rceil$ constitutes a limit value for the current scheduling error over which the required bandwidth is saturated to U_{max}. The control law is visualised by the dashed line in Figure 4, where the budget assignment is shown for various values of ϵ_j, and for $H_{j+1} = 5, P = 10, N = 10$. Clearly, if the task receives more bandwidth than required in Equation (8), the robustness with respect to prediction errors is increased, and so is the probability for ϵ_j to remain in the stability region. This is consistent with the requirement in Equation (5), which establishes a *minimum* probability π. This motivates the application of reclaiming techniques as discussed in Section 5.2.

As shown in Figure 3, the bandwidth *required* by a task controller is not necessarily *granted* by the supervisor, which enforces the schedulability constraint in Equation (1) at all times. These two contrasting needs are reconciled by the notion of *minimum guaranteed bandwidth* $\overline{B} \leq U_{max}$, and the corresponding *minimum guaranteed budget* $\overline{Q} = \frac{\overline{B}}{P}$: when a task controller issues a budget request lower than $\overline{Q}$, the request is granted by the supervisor; however, if the request is higher than $\overline{Q}$, then the supervisor grants at least a budget of $\overline{Q}$, and exactly $\overline{Q}$ in the worst-case. In the next subsections, we will discuss the closed loop guarantees that each task can be offered depending on the value of the minimum guaranteed budget.

4.2.1 *Conditioned probabilistic guarantee.* If the budget assignment of a task is governed by the law presented in Equation (8), then the controlled task meets requirement in Equation (6) as formally stated in the following result (see the appendix for the proof).

THEOREM 4.1. *Consider the system defined in Equation* (4)*. If the following conditions hold:*

(*1*) *the predictor produces an estimate* H_{j+1} *of* $\mathcal{C}_{j+1}$ *such that*

$$\mathbf{Pr}\{\mathcal{C}_{j+1} \leq H_{j+1}\} \geq \pi; \tag{9}$$

(*2*) R *satisfies* $R \leq E_j \triangleq N - \left\lceil \frac{H_{j+1}}{PU_{max}} \right\rceil$;

(*3*) J_{j+1} *is granted at least the budget* $Q(\epsilon_j, H_{j+1})$ *as defined by Equation* (8)*;*

then, the requirement in Equation (6) *is fulfilled.*

Another way to look at the above statement is that, as long as ϵ_j is below the point the control law saturates (point E_j in Figure 4), it is reduced in the stability region with at least a probability equal to the one that the predictor succeeds. This point E_j changes for each job, and it has to always be greater than the upper bound R of the attractivity basin. Based on the result above, a more conservative

prediction (i.e., a larger H_j) allows for increasing the probability π, but it also causes the increase of the required bandwidth.

Due to the above result, the just introduced control law is referred to as PDNV controller (Probability of Deadline Non-Violation).

It is clear that this property is valid only as long as the supervisor grants the budget required by the task controller. As an example, the continuous line of Figure 4 shows the actual budget assignment that may result, with a request indicated by the dashed line, after the supervision action with a minimum budget of $\overline{Q} = 2.5$. In the worst-case in which no more than $\overline{Q}$ is available for the task, the budget assignment is saturated to $\overline{Q}$ after the point $\epsilon_j = M_j$ where the requested budget $Q(\epsilon_j, H_{j+1})$ intercepts $\overline{Q}$:

$$M_j \triangleq N - \left\lceil \frac{H_{j+1}}{\overline{Q}} \right\rceil . \tag{10}$$

It is not difficult to account for such supervisory action in Theorem 4.1, as shown in the following result (see the appendix for the proof).

THEOREM 4.2. *If the following conditions hold:*

(1) $\mathbf{Pr}\{\mathcal{C}_{j+1} \leq H_{j+1}\} \geq \pi$;

(2) the minimum budget $\overline{Q}$ *configured for the task within the supervisor satisfies*

$$\overline{Q} \geq \frac{\sup_j H_j}{N - R}; \tag{11}$$

then, the control law in Equation (8) *fulfils the requirement in Equation* (6).

From a practical standpoint, the $\sup_j H_j$ quantity in the just stated result is the critical value to obtain: recalling that H_j is supposed to verify Equation (9), it is reasonable to use, in such expression, an estimate of the π percentile of the computation times distribution, as available from previous benchmarking runs of the real-time task. Such approach will find its validation in our experimental results shown in Section 7. However, in strongly dynamic systems in which the $\mathcal{C}_j$ distribution is expected to be time-varying, the sup operator in the above result would amount to requiring a value of H_{j+1} representing a percentile estimation of $\mathcal{C}_{j+1}$ under worst-case conditions. While such a value provides very robust guarantees to the real-time task, we envision the possibility to configure the supervisor with a value of $\overline{Q}$ which is actually lower, in order to increase the average saturation level of resources. Clearly, the guarantees provided to the application would be correspondingly weakened.

4.2.2 *Deterministic guarantee.* To complete the picture, we need to specify the behaviour of the system when the scheduling error falls, at some point in time, outside of the attractivity basin. Here, our requirement formalised in Equation (7) comes into play: when the scheduling error is greater than R then it *has* to return below R in a finite number of steps *and* its maximum value has to be limited. To show a condition that guarantees this property, we need to introduce the maximum error incurred by the predictor: $\rho \triangleq \sup_j \frac{c_j}{H_j}$, assumed to be > 1. We can then state the following result, whose proof is in the appendix.

THEOREM 4.3. *Consider the system* (4)*, controlled by the control law in Equation* (8) *with a predictor characterised by* ρ*. Let* ϕ_V *denote the worst-case moving average of the computation times over* V *samples:* $\phi_V \triangleq \sup_h \frac{1}{V} \sum_{i=1}^{V} c_{h+i}$*. If the following conditions hold:*

(*1*) *for some* $V \in \mathbb{N}$*,*

$$\overline{Q} > \frac{\phi_V}{N-1}; \tag{12}$$

(*2*) $0 < R \leq \inf_j M_j$*, where* $M_j = N - \left\lceil \frac{H_{j+1}}{\overline{Q}} \right\rceil$;

then, the requirement in Equation (7) *is satisfied with:*

$$\begin{cases} L = \left\lceil \frac{(\rho-1)N+1-R}{N-1-\frac{\phi_V}{\overline{Q}}} \right\rceil \\ \epsilon^{MAX} = \left\lceil \frac{V\phi_V}{\overline{Q}} \right\rceil + N(\rho-1) - (N-1). \end{cases} \tag{13}$$

In this theorem, we require a level of guaranteed bandwidth sufficient to outweigh any "moving average" of the computation times over a time horizon of length V, and this value is generally greater than the mean value of the process. Therefore, our condition is generally stronger than the classical stochastic stability condition (commonplace in queueing theory [Kleinrock and Gail 1976]), but it also leads to stronger guarantees. Indeed, not only do we guarantee that the scheduling error will have a finite mean, but also that it remains bounded.

4.2.3 *Unconditioned probabilistic guarantee.* Now we are in the position to consider jointly the probabilistic and deterministic results shown above, in order to provide a bound to the unconditioned probability that the scheduling error resides in the stability region $\epsilon_j \leq 0$. This way, the limitations of Theorem 4.2 may be overcome by considering also Theorem 4.3.

In order to carry out the analysis, we assume that the process $\mathcal{C}_j$ is independent (but non-necessarily identically distributed), and that H_j are deterministic functions of j. This is possible, for instance, if the distributions of $\mathcal{C}_j$ can be known or estimated by the predictor.

We are now able to state the following result, whose proof is in the appendix.

THEOREM 4.4. *Consider the system in Equation* (4)*, controlled by the control law in Equation* (8)*. Assume that conditions of Theorems 4.2 and 4.3 hold, that* $\mathcal{C}_j$ *is an independent process, and that* $\forall j$*,* $M_j \geq 0$*. Then:*

$$\mathbf{Pr}\{\epsilon_j \leq 0\} \geq \frac{\pi}{1 + L(1-\pi)} \tag{14}$$

where L *is the bound defined in Equation* (13)*.*

The bound introduced above can clearly be refined leveraging a better knowledge of the computation times. For example, knowing their exact probability distribution allows for a numerical analysis based on the resolution of a Markov chain with a

state for each possible ϵ_j value. However, Equation 14 establishes in closed form clear limitations and minimal guarantees on the performance of our controller, which may be conveniently leveraged at design time.

Finally, note that, if the process $\mathcal{C}_j$ were correlated, and if we had a model describing the correlation (e.g., an ARMA or an ARMAX process), then we could generalise the analysis proposed in this section. In this case, a punctual value of the scheduling error would not be sufficient anymore to represent the state of the MC, but we would need to augment the state to account for the past history of $\mathcal{C}_j$.

5. GLOBAL SUPERVISOR

The main objectives of the supervisor can be summarised as follows.

First, when the task controllers implementing the adaptive reservation control loops formulate bandwidth requests violating the constraint in Equation (1), the supervisor enacts a *compression* to reduce the total bandwidth within the limit U_{max}. Such condition is possible due to the independence among the various control loops. Even when the admission control is based on conservative estimates for $\overline{Q}$, in our framework it may still happen that, temporarily, a task exhibits unforeseen spikes of computation times and/or the scheduling error exhibits unforeseen delays, therefore the associated task controller would submit to the supervisor a request which is greater than $\overline{Q}$.

Second, when the total request of the tasks is below U_{max}, or when a task finishes a job without using all the allocated budget, a reclaiming mechanism redistributes the unused budget to the other tasks. This expansion has the evident effect of reducing the scheduling error. Therefore, it alleviates potential problems arising from wrong predictions of the computation times, and increases the robustness of the controllers.

In the rest of the section, we will show how the proposed supervisor achieves these goals. Then, we will briefly discuss some important implementation details related to the interaction between AR and the reclaiming mechanism, which introduce latency in actuating the control decision.

5.1 Compression

In managing overload conditions (i.e., the bandwidth requests B_i exceed U_{max}), the supervisor, in order to not violate Equation (1), grants to each task at least its minimum guaranteed bandwidth if requested, i.e., a value of $B_i^m \triangleq \min\left\{\overline{B}_i,\, B_i\right\}$. Then, it distributes the available bandwidth $U^a \triangleq U_{max} - \sum_i B_i^m$ according to some system-wide policy, accounting for a set of weights $\{w_i\}$ representing the relative importance of applications, obtaining the granted bandwidth figures B_i^g :

$$B_i^g = B_i^m + f_i\left(U^a, \left\{B_j - B_j^m\right\}_j, \{w_j\}\right) \tag{15}$$

with the constraint that $\sum_i f_i = U^a$. We do not mandate any particular function to use as $f_i(\cdot)$, for example a simple linear decrease of the additional bandwidth from $(B_i - B_i^m)$ to 0, which starts with a gradient oriented along the vector of weights may be used (as done in the experiments shown in Section 7).

5.2 The reclaiming mechanism

The expansion mechanism was designed considering two important requirements. The first requirement is to have a very reduced latency between the time the spare bandwidth becomes available and the time it is received by the tasks. The second one is to fairly distribute the extra bandwidth between the tasks, so as to increase the robustness of all controllers.

To this purpose, we developed SHRUB (Shared Reclamation of Unused Bandwidth), a variant of GRUB (Greedy Reclamation of Unused Bandwidth [Lipari and Baruah 2000]), which in turn is based on the CBS [Abeni and Buttazzo 1998]. GRUB is known to be very responsive in reclaiming the unused bandwidth, however it does it in a greedy way. SHRUB, instead, performs a fair redistribution of the unused bandwidth. Before describing SHRUB, it is useful to quickly overview GRUB.

5.2.1 *GRUB in a nutshell.* As for the CBS algorithm, in GRUB each task is assigned a reservation (Q_i, P_i). Like in the CBS algorithm, every reservation has a *current budget* q_i and a *scheduling deadline* d_i^s, both initialised to 0. For reservations whose corresponding task is not active, GRUB also defines the *idling instant* $I_i = d_i^s - \frac{q_i}{P_i}$.

In GRUB we introduce the concept of *state* of a reservation. At any time, a reservation can be in one of the following states: inactive if the corresponding task is not active and its idling instant is in the past; activeContending if the corresponding task is active (i.e., it has a pending job); activeNonContending if the corresponding task is not active and the idling instant is in the future.

Initially, all reservations are in the inactive state. For a given reservation we have the following state transitions:

(1) when a task is activated at time t : a) if the reservation is inactive it changes to the activeContending state, the budget is updated to $q_i = Q_i$ and the scheduling deadline is set to $d_i^s = t + P_i$; b) if the reservation is in the activeNonContending state, it moves to the activeContending state maintaining its budget and deadline unchanged;

(2) when a job completes, if there are no more pending jobs, the reservation changes its state to a) activeNonContending if the idling instant is in the future; b) to the inactive state if the idling instant is in the past or at the current time;

(3) when the budget is depleted, following the CBS rule, it is immediately recharged to $q_i = Q_i$ and the scheduling deadline is postponed to $d_i^s = d_i^s + P_i$;

(4) finally, a reservation remains in activeNonContending state until its idling instant, then it moves to the inactive state.

All reservations in the activeContending and in the activeNonContending state are said to be *active*. The sum of the bandwidths of the active reservations $B_{act}(t)$ represents the total amount of used bandwidth in the system at a given time. $B_{act}(t)$ is increased every time a reservation goes from inactive to activeContending, and is decreased every time a reservation enters the inactive state.

The main idea behind GRUB is that the bandwidth $(U_{max} - B_{act}(t))$ is not used and can be re-distributed among needing reservations. The re-distribution is performed by acting on the rule to update the budgets.

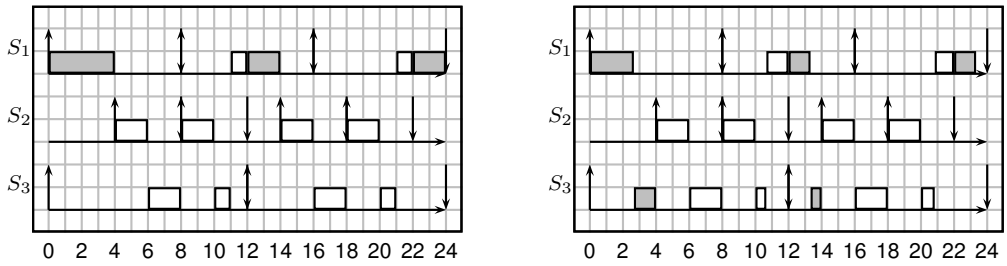

Fig. 5. Example of schedule with GRUBand with SHRUB. GRUB assigns all the excess bandwidth to S_1, while SHRUB fairly distributes it between S_1 and S_3

In GRUB all the reclaimed bandwidth is greedily assigned to the current executing reservation. The rule[1] for updating the budget is:

$$dq_i = \begin{cases} (1 - U_{max} + B_{act})dt & \text{for executing reserv.} \\ 0 & \text{for non exec. reserv.} \end{cases} \quad (16)$$

5.2.2 *The SHRUB algorithm.* GRUB is a greedy algorithm because all excess bandwidth is assigned to the executing reservation. Therefore, in certain pathological cases it may happen that one reservation gets all the spare bandwidth, and the others get nothing.

An example is shown in the left side of Figure 5. Consider three reservations S_1, S_2, and S_3, with bandwidth $B_1 = 0.25$, $B_2 = 0.5$, and $B_3 = 0.25$, and with periods $P_1 = 8$, $P_2 = 4$, and $P_3 = 12$, respectively. We assume that $U_{max} = 1$. Also, for the sake of clarity, and to highlight the properties of GRUB, we assume that reservations S_1 and S_3 are always activeContending, while reservation S_2 serves a sporadic task with constant execution time equal to $C_2 = 2$ and arrivals at $t = 4$, $t = 8$, $t = 14$ and $t = 18$.

At time $t = 0$, only S_1 and S_3 are active, hence $B_{act} = 0.5$. S_1 is the reservation that executes, because its deadline is the earliest, and its budget is decreased at a rate of B_{act}. Hence, its budget is exhausted at time $t = 4$ (instead of t_2 as in the CBS algorithm), and its deadline is postponed to $d_1 = 16$. The reclaiming intervals are highlighted by a grey colour in the execution time. At this time, reservation S_2 is activated (the sporadic task arrives at $t = 4$), so $B_{act} = 1$. The active bandwidth remains equal to 1 until $t = 12$, when the idling instant for S_2 is reached, and the reservation becomes inactive. At this point, reservation S_1 can execute decreasing its budget at a rate $B_{act} = 0.5$, and thus its budget is exhausted at time $t = 14$.

In this example all excess bandwidth is reclaimed by S_1, while S_2 does not get any advantage. It has been shown with extensive simulations [Lipari and Baruah 2000] that, if tasks are independent and computation times uniformly distributed, in the long period the excess bandwidth is distributed among all reservations in proportion to their bandwidth B_i. However, such fair distribution cannot be controlled.

To overcome this problem, we present the SHRUB algorithm that fairly distributes the unused bandwidth among all active reservations in proportion to their weights.

[1]Due to space constraints, we cannot report the full rationale for the updating rule. The interested reader can refer to [Lipari and Baruah 2000] for further details.

In SHRUB, every reservation is assigned a weight $w_i \geq 0$.

Then, SHRUB is almost identical to GRUB, the only difference being in the rule to update the budgets. Instead of using Equation (16), it uses a more fair rule:

$$dq_i = \begin{cases} \left(-1 + (U_{max} - B_{act})\frac{w_i}{W_{act}}\right) dt & \text{for exec. reserv.} \\ (U_{max} - B_{act})\frac{w_i}{W_{act}} dt & \text{for non exec. reserv.} \end{cases} \quad (17)$$

where $W_{act}(t) \triangleq \sum_{act} w_i$ is the sum of the weights of all active reservations. Note that, in case one or more weights is null, the budget update rule must only be applied if $W_{act}(t) > 0$, otherwise $dq_i = -dt$ for the currently executing server, and $dq_i = 0$ for the others.

To show how this new rule enables a fair distribution of the extra bandwidth, consider again the example of Figure 5. Suppose that all reservations are assigned equal weights $w_i = 1$. By scheduling the same system with SHRUB we obtain the schedule shown in the right side of the figure. Notice that S_1 now executes for less than in the previous case, and part of the excess bandwidth is now assigned to S_3.

It can be shown that the introduction of this new feature in SHRUB does not invalidate the basic properties of GRUB. In particular, with SHRUB:

— the *temporal isolation property* holds, that is each reservation is always allocated at least the reserved amount of execution $Q_i = B_i P_i$ every period P_i;

— the *hard schedulability property* holds, which states that if a hard real-time periodic task with worst case computation time C_i and period T_i is assigned a reservation with $Q_i \geq C_i$ and period $P_i \leq T_i$, then it will respect all its deadlines;

— no assumption is necessary on the task model; both algorithms can be used for periodic, sporadic and aperiodic tasks.

In addition, by properly assigning the weights, it is possible to obtain a range of possible behaviours with respect to the reclaiming. For example, by assigning a null weight, the reservation simply does not receive any extra bandwidth. With a very large weight, instead, almost all extra bandwidth will be assigned to the corresponding reservation. Therefore, it is possible to precisely control the extra bandwidth allocation by simply assigning the appropriate weights to the reservations.

A comprehensive description of SHRUB is out of the scope of this paper and can be found in [Baruah et al. 2008].

5.3 Combining feedback and reclaiming

Feedback scheduling can be applied on top of any resource reservation scheduler, including the ones that provide expansion through reclaiming. However, we must ensure that the two mechanisms do not interfere with each other, and instead work together by re-enforcing their good properties.

The first condition regards the control goal. In certain cases, the control goal is to maintain the scheduling error around 0, or more generally, to control both the upper and the lower bound of the scheduling error. This implies that the finishing time of the task is always within a certain interval of time. Therefore, controlling both the lower and the upper bound of the scheduling can be used to control the finishing time jitter of a task.

In such cases, while the control law tends to assign small budgets to prevent the task from completing too early, the reclaiming algorithm tends to reclaim unused bandwidth thus increasing the reservation budget and nullifying the effects of the feedback. When using SHRUB, it is possible to assign to these tasks an AR with a weight $w_i = 0$. Thus, this task will never receive any extra bandwidth, and the control strategy will not be disturbed.

Instead, a controller whose goal is to control only an upper bound of the scheduling error (like the one used in this paper) is perfectly compatible with reclaiming policies. Furthermore, reclaiming can greatly reinforce the control action, since they are in the same direction: in case of positive scheduling error, or in case of increasing execution time prediction, the controller will increase the reservation budget. Such command will be reinforced if there is spare bandwidth in the system by the reclaiming policy. Therefore, reclaiming increases the robustness of the controller, for example by making the control performance less sensible to prediction errors. Tasks should be assigned a SHRUB weight that is larger for more sensible tasks.

5.4 Actuation latency

A particular care must be taken in finding a correct timing between the bandwidth request of the adaptive reservations and the reclaiming mechanism. Indeed, Equations (16) and (17) assume that the reserved budget is constant over time. Adaptive reservations, instead, modify the budget according to the control law at the end of every job of the task. In SHRUB, reservation budgets can be modified in two cases: when the reservation is inactive, or at its *idling time* (i.e., when it is about to become idle). This means that a control action increasing the current budget, that is performed when the reservation is activeContending or activeNonContending, must be delayed until one of the two previous conditions is verified. This interval of time is usually very small, and in the worst-case it is equal to the maximum server-period among reservations in the system: $P^{max} \triangleq \max\{P_i\}$. Note that, in the scheduler we used, it is possible to limit the maximum value of the reservation period for all reservations admitted in the system, by means of a system-wide configuration file. See [Cucinotta 2008] for details.

However, the effect of this delay on control design is merely the need for some over-provisioning of the required budget, as compared to the figures appearing in Section 4. In fact, it is not difficult to show that, in case of an increase of the current budget $Q_{i,j+1} > Q_{i,j}$, and in the worst-case of a maximum actuation delay of P^{max}, Equation (4) may be rectified as follows:

$$\epsilon_{i,\,j+1} \leq S(\epsilon_{i,\,j}) + k_i + \left\lceil \frac{c_{i,\,j+1} - k_i Q_{i,j}}{Q_{i,j+1}} \right\rceil - N_i, \tag{18}$$

where $k_i \triangleq \left\lceil \frac{P^{max}}{P_i} \right\rceil$. Such expression has been obtained under the simplifying assumption that $c_{i,\,j+1} \geq k_i Q_{i,j}$. For example, the direct consequence on the control law in Equation (8) is that it should be modified as:

$$\tilde{Q}(\epsilon_j,\, H_{j+1}) = \begin{cases} \frac{H_{j+1} - k_i Q_{i,j}}{N - S(\epsilon_j) - k_i} & \text{if } \epsilon_j \leq \tilde{M}_j \\ \overline{Q} & \text{otherwise,} \end{cases} \tag{19}$$

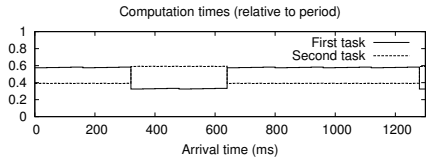

Fig. 6. Computation times for the two tasks.

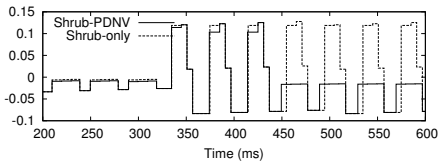

Fig. 7. Scheduling errors experienced by the second task. Each scheduling error value is represented as a horizontal step, starting from the finishing time of the job it relates to, up to the next job finishing time (horizontal steps are only drawn for ease of reading).

with $\tilde{M}_j \triangleq N - \left\lceil \frac{H_{j+1} - k_i Q_{i,j}}{\overline{Q}} \right\rceil$. Similarly, other presented results may be easily reformulated accounting for the delay.

6. ADVANTAGES OF COMBINING FEEDBACK AND RECLAIMING

Before providing full experimental evidence of the advantages of our approach, we discuss here how the feedback scheduling mechanism can improve the action of resource reclaiming. Clearly, if the system is overloaded, the reclaiming cannot operate. In this section, we show on a simple example that, if the computation requirements of the task change and if the periods are different, the reclaiming mechanism does not offer an acceptable performance even in the case of underloaded system. The example has been developed using an open-source adaptive reservation simulation framework (ARSim) [2], able to simulate the evolution of the system defined in Equation (4) along with the adaptive reservations and the global supervisor (as depicted in Figure 3). In particular, we implemented in ARSim the controller described in Section 4 and the supervisor described in Section 5.

The example is composed of two tasks τ_1 and τ_2. *The periods of the tasks are quite different from each other*, namely $T_1 = 40ms$ and $T_2 = 10ms$. The server periods P_1 and P_2 were chosen very small so that their impact was not relevant for the purposes of this simple example. The workload is structured to require, on the average, a bandwidth of 60% for the first task, and of 40% for the second one. Therefore, it is quite natural to tune the minimum guaranteed bandwidths for the two tasks to achieve the same relative ratio. However, there is a time window (jobs with activation between the 250^{th} and the 620^{th} millisecond, as zoomed in Figure 6) in which the workload dynamically changes so that the requirements of the two tasks are basically swapped.

During this time window, the system is still schedulable, but, as shown by the simulation, the dynamic reclaiming performed by the SHRUB algorithm alone is not sufficient to compensate for the mismatch between the bandwidth values statically guaranteed to applications and the actual workload. In fact, the large overallocation of bandwidth performed for the first task leads it to finish very early (its scheduling error is not shown in the pictures), so that until the next activation the CPU is entirely granted to the second task. Unfortunately, up to this moment, the

[2]More information is available at: `https://gna.org/projects/arsim`.

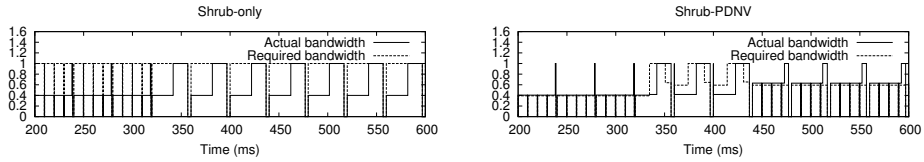

Fig. 8. Required and actually granted bandwidth when using SHRUB alone without feedback (left) and together with the PDNV controller (right).

second task has already completed two periods, with a bandwidth allocation lower than required, therefore there are deadline violations (highlighted by the positive scheduling error values of the dashed line in the scheduling error temporal evolution in Figure 7). On the other hand, the subsequent two jobs of the second task before the reactivation of the first task are completed with a 100% CPU allocation, so they complete much earlier (the negative scheduling error values of the same line). The problem is that for the second task, during this temporary mismatch between the static allocation and the actual workload, only two periods out of every sequence of four take actually advantage of the SHRUB dynamic bandwidth reclaiming, while the other two periods suffer of the same problems that would occur with a statically fixed bandwidth allocation.

When we use a PDNV controller together with SHRUB, after a few jobs, the controllers steer the bandwidth requirements of the two tasks to match the actual workload, as shown in the right side of Figure 8. Here, we show the bandwidth required by the controller versus the actual granted bandwidths after the compression performed by the supervisor. Also, compare this temporal behaviour with the case of SHRUB only, shown on the left side of the figure. This results in a scheduling error that, after the adaptation of the controller, returns to respect the deadline for every job (continuous line).

7. EXPERIMENTS

This section validates the techniques shown in Sections 4 and 5, by showing experimental results gathered by running real applications using an implementation of the proposed techniques in the Linux OS. More specifically, we used the AQuoSA [Cucinotta et al. 2008] (Adaptive Quality of Service Architecture) open-source real-time scheduler for Linux.

The open-source AQuoSA architecture is composed of:

—a patch to the Linux kernel that exposes scheduling-related events to dynamically loadable modules;

—a dynamically loadable module implementing an EDF-based real-time scheduler;

—a user-space library that may be used by applications to exploit the real-time services of the patched real-time kernel;

—a QoS management library that embeds a set of QoS control laws and prediction algorithms of general use, and allows for the realisation of application-specific bandwidth allocation policies and/or predictors by applications themselves.

The AQuoSA scheduler has been modified so as to implement the SHRUB soft

	τ_1	τ_2
Period (T_i)	$15ms$	$60ms$
Frequency ($1/T_i$)	$66.6Hz$	$16.6Hz$
Mean of $\{c_i, j\}$	$3ms$	$27.5ms$
Standard deviation of $\{c_i, j\}$	$0.5ms$	$3.65ms$
Minimum of $\{c_i, j\}$	$1.25ms$	$8.7ms$
Maximum of $\{c_i, j\}$	$8.3ms$	$38.25ms$
83.33^{rd} percentile of $\{c_i, j\}$	$3.42ms$	$30.7ms$

Table II. Statistics on the workload of the two MPEG decoding tasks.

reservation scheduler as described in Section 5, and the QoS management library has been extended to include an implementation of the QoS control technique introduced in Section 4. Further details about the architecture may be found in the project website[3]. The experiments that follow have been performed on a Linux Kernel 2.6.22 series, with support for high-resolution timers.

7.1 Prediction algorithm

The prediction algorithm we used in our experiments is a general purpose one that estimates the π percentile of H_{j+1} assuming it exhibits a distribution similar to the last observed k computation times $C_{j,k} \triangleq (c_j, \ldots, c_{j-k+1})$. Therefore, defining $h \triangleq \lceil k(1-\pi) \rceil + 1$, the predictor computes the h^{th} maximum of the samples. For example, with $k = 12$, the value of H_{j+1} corresponding to $\pi = 100\%$ is achieved by taking of course the first maximum of $C_{j,k}$; for $\pi \in [91.\overline{6}, 1[$ it takes the 2^{nd} maximum, for $\pi \in [83.\overline{3}, 91.\overline{6}[$ the 3^{rd} one, and so on. The rationale behind this idea is to produce a value H_j that is an upper-bound for c_j with a probability approximately equal to the percentile.

7.2 Synthetic real-time application

First of all, the results obtained through simulations in Section 6 have been confirmed by performing some experiments. In other words, the feedback scheduler implementation presented above has been used to highlight the advantages of combining feedback and reclaiming. To this purpose, we used a synthetic application adhering to the model of periodic real-time task as defined in Section 2. For each job $J_j j$, executes for a time c_j read from an input file (the application needs is tuned in order to build a mapping of expected execution times to number of repetitions of an internal loop that performs some algebraic operations).

We ran concurrently two instances of the synthetic application, running at activation rates of $16.66Hz$ and $66.66Hz$, using as trace files the execution times needed for decoding MPEG videos, as measured during real runs of the `FFmpeg`[4] software. In all the runs the server period for the tasks has been set to $3ms$. The relevant timing parameters of the two tasks are summarised in Table II.

The experiment has been repeated with three different configurations: first, we used only the feedback-based QoS control loop to allocate the job-by-job bandwidth allocation for the two tasks, and the underlying scheduler has been configured so

[3]More information is available at: `http://aquosa.sourceforge.net`.
[4]More information is available at: `http://www.ffmpeg.org`.

as to provide a hard resource reservation policy (i.e., a policy that gives each task its allocated bandwidth without any reclaiming). Then, we repeated the experiment without feedback-based QoS control, by using only the SHRUB soft resource reservation scheduling policy, at varying configurations for the static bandwidth allocation for the two tasks. Finally, we ran the applications with the both the PNDV controller and the SHRUB based supervisor enabled.

The results of the experiment are shown in the left side of Figure 9. The picture reports on the horizontal and vertical axes the experimental probabilities of respecting the deadline achieved, respectively, for the periodic task with lower and higher period. Therefore, each point in the graph represents the experimental probabilities of achieving a scheduling error less than or equal to the deadline, for the two tasks. The "ideal" region that result on such a plot is a point on the upper-right region, close to a value of 1 for the PDNV of both tasks.

When applying the feedback alone (single point tagged with "FB Only"), the tasks realise PDNV values that are close to the percentiles configured into the QoS controllers, both set to $83,33\%$. Though, the particular overload characteristics of the task set causes the second task to exhibit a lower PDNV, with a value of 70%.

When using the SHRUB scheduling policy alone without feedback-based scheduling (points on the curve tagged as "SHRUB Only", obtained at varying values of the minimum guaranteed bandwidths), we achieved a slightly better PDNV value for the second task (close to 70%), but a significantly higher PDNV value for the first task. This is to be expected, because whilst the PDNV controllers are run with hard resource reservations (therefore the residual bandwidth is not reclaimed if a job terminates before the foreseen c_k has been consumed), the SHRUB scheduling policy is capable of using the full processing power of the processor, thus it achieves better average performances. Also, it is shown that, playing with the static allocation for the two tasks, it is possible to increase the PDNV value for one of the tasks, but at the cost of a very steep decrease in the PDNV value of the other task.

Finally, when using the proposed technique, constituted by the QoS control loop and the SHRUB soft reservation scheduling policy, the points on the curve tagged as "FB and SHRUB" are achieved. In this case, the PDNV values achieved are higher, and they generally outperform the performance achieved by the SHRUB technique alone, for each configuration of minimum guaranteed bandwidths.

This proves that not only does the combination of the two techniques achieve a good system performance, independently of the initial configuration of the system in terms of minimum guaranteed bandwidths (self-tuning), but also that the achievable performance is significantly higher than the one that is achievable when using either the feedback alone or the SHRUB scheduling policy alone.

Basically, feedback-based scheduling is capable of adding adaptivity and self-tuning to SHRUB, and SHRUB is capable of adding robustness with respect to prediction errors to the QoS controllers used within feedback-based scheduling.

After verifying the advantages offered by the combination of reclaiming and feedback scheduling the theory of QoS control loops described in Section 4 has been verified. To this purpose, we ran an experiment similar to the previous one, where the two tasks have been run with a hard resource reservation policy and feedback-based dynamic resource allocation. We used the same QoS controller configuration

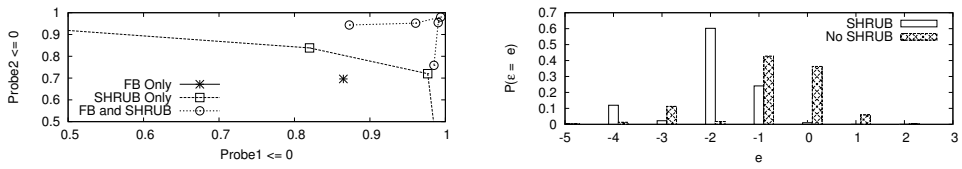

Fig. 9. Probability of Deadline Non-Violation for two synthetic real-time tasks (left). Scheduling error PMF for `streamer` with and without SHRUB (right).

of the previous subsection, with both percentiles tuned on the 83.33% values. In this case, we tuned the experiment by performing a prior benchmarking phase in which the minimum value for the guaranteed budget respecting Equation (12) in Theorem 4.3 has been computed. Then, we ran the experiment a second time by using these minimum guaranteed budgets, increased relatively by 1% (the execution times never repeat exactly equal). The QoS control loops exhibited PDNV values of 95.97% and 85.59% for the two tasks, and, examining the collected job-by-job scheduling error evolution, we found that the maximum number of jobs for which the scheduling error was outside of the target region ($\mathcal{E} \leq 0$) was 2 for both tasks, perfectly matching the theoretical condition of Theorem 4.3.

7.3 Experiments on a real application

After performing some experiments with synthetic real-time applications, a real multimedia application has been used for testing the feedback scheduler implementation. The selected application is a video encoder/streamer based on `FFmpeg`. Such application (referenced to as `streamer` from now on) grabs frames from a `video4linux2` device, encodes them in MPEG4 [m4v 2004] video (by using the `FFmpeg` `libavcodec` library) and streams the encoded frames to remote clients by using the RTP protocol [Schulzrinne et al. 2003].

First of all, a single instance of `streamer` has been used to verify that there is a match between the guarantee provided by Theorem 4.4 and the experimental evidence. To this end, a trace of the execution times of `streamer` has been recorded and used to dimension the feedback parameters according to Theorem 4.4. According to such theorem, a feedback controller with $P = T/6$, $\overline{Q}/T = 0.78$ using a predictor which discards 2 samples every 12 is guaranteed to respect a task deadline ($\epsilon < 0$) with a probability ≥ 0.83. This bound has been verified by scheduling `streamer` with a controller characterised by the parameters mentioned above: when SHRUB reclaiming is not used, the probability $P\{\epsilon \leq 0\}$ to respect a deadline has been measured as 0.931, while with SHRUB $P\{\epsilon \leq 0\} = 0.9996$. Since both these values are larger than 0.83, Theorem 4.4 is verified.

The Probability Mass Function (PMF) of the measured scheduling error is displayed in the right side of Figure 9, which also shows the advantages provided by SHRUB reclaiming. Such advantages are more visible in a second experiment, where two instances of `streamer` have been run simultaneously (encoding smaller frames, so the requested budgets are smaller), grabbing videos from two webcams at $10fps$ and $30fps$. Therefore, the system is composed by 2 tasks τ_1 and τ_2 with periods $T_1 = 100ms$ and $T_2 = 33.3ms$; the server periods are $P_1 = T_1/6$ and

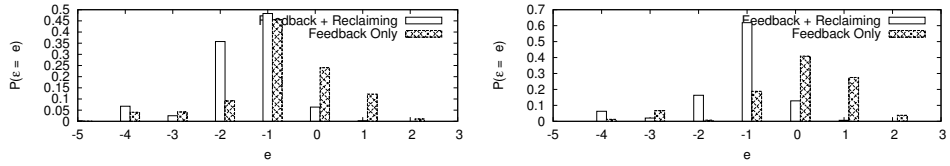

Fig. 10. PMF of the scheduling errors for the first `streamer` instance (task τ_1, on the left) and the second one (τ_2, on the right), when using feedback with and without reclaiming.

$P_2 = T_2/6$.

The experiment has been repeated by first using the feedback mechanism alone, and then by using feedback + reclaiming. Figure 10 compares the PMF of the scheduling error for the two tasks when the feedback mechanism is used without any form of reclaiming, and when reclaiming is used together with the feedback. By looking at the two figures it is possible to notice that the reclaiming mechanism allows to reduce the scheduling error experienced by the two tasks. However, it is difficult to quantify the improvements introduced by reclaiming. To compare the results through a more objective and quantitative metrics, the experimental probabilities $P\{\epsilon_i \leq 0\}$ to respect the deadline have been measured.

When using feedback-based scheduling only (without reclaiming), such probabilities have been experimentally measured as $P\{\epsilon_1 \leq 0\} = 0.867798$ and $P\{\epsilon_2 \leq 0\} = 0.684411$. On the other hand, when also the SHRUB reclaiming was activated together with the feedback, the measured experimental probabilities increased to $P\{\epsilon_1 \leq 0\} = 0.998301$ and $P\{\epsilon_2 \leq 0\} = 0.993669$.

By looking at these numbers it is possible to understand that the predictor is probably underestimating the execution times for task τ_2; as a result, when using the feedback alone, τ_2 probability to respect the deadline is low. On the other hand, the reclaiming mechanism is able to correct the under-estimation by increasing the probability to 0.99. Hence reclaiming makes the feedback mechanism more robust.

Therefore, the addition of dynamic reclaiming to our feedback-based scheduling algorithm allows for an enhancement of its performance, as expected. In fact, our QoS control metrics is compatible with a soft reservation resource allocation, being based on requiring a minimum value for the probability of deadline non-violation. From a dual perspective, we can also conclude that the addition of a feedback-based control loop to a real-time system using dynamic reclaiming allows for a self-tuning of the scheduling parameters of the various tasks that also accounts for the dynamic variability of the workload at run-time.

In the next experiment, we will show also that the combination of feedback scheduling and reclaiming is capable of outperforming any possible static allocation of bandwidth for a task set with well-known workload type.

8. RELATED WORK

One of the basic motivations for this work is the high variability exhibited by the computation time of multimedia applications. This problem has been deeply documented [Hughes et al. 2001; Isović et al. 2005]. Many authors propose adaptivity as a means to increase the system robustness to workload fluctuations.

Most of these proposal advocate an application level adaptation. The idea is that in response to time-varying application requirements and availability of shared resources, the behaviour of the applications is changed at run-time to make their requirements fit in the instantaneous resource availability. A first line of research has proposed application level adaptation for general real-time applications. An incomplete list of papers of this kind includes [Tokuda and Kitayama 1993; Nakajima 1998; Brandt and Nutt 2002]. Other authors have specialised this approach looking at multimedia applications. For instance in [Isovic and Fohler 2004], the authors propose an optimal strategy that skips frames maintaining high levels of quality and respecting the timing constraints. In [Wüst and Verhaegh 2004; Wüst et al. 2005] the authors take a different approach: frames are not skipped but the quality of level of the stream can be changed on a frame by frame basis operating on such parameters as the resolution or the number of layers. A similar idea is presented in [Hentschel et al. 2001], where the authors propose scalable quality video as a means to achieve adaptive behaviours. In [Combaz and Strus 2008] the authors synthesise a QoS manager that takes as input the degree of criticality of the different deadlines of the tasks and adapts their quality level based on the distribution functions of the computation times that are collected on-line. In [Lan et al. 2001] the authors use a predictor that allows them to foresee the workload peaks and reduce them operating on the quality level of the stream. Although a very effective technique, application level adaptation relies on a particular structure for the applications (e.g., the presence of discrete quality options that can be switched on-line). In this paper we adopt *a complementary strategy* that performs the adaptation operating on the scheduling parameters, and ultimately on the resource allocation. The co-existence of two levels of adaptation (at the scheduling level and at the application level) has been studied in [Abeni and Buttazzo 2001].

The idea of dynamically adapting scheduling parameters based on some observed value (also known as *feedback scheduling*) has been investigated in the past [Corbato et al. 1962]. A recent innovation has been to establish a link between the adaptation of the scheduling parameters and the real-time performance of the application. In [C. Lu and Son 2002] the authors propose to adjust the deadlines in a Earliest Deadline First scheduler based on the frequency of occurrence of deadline misses. We believe that controlling the frequency of deadline misses (or more precisely the probability of a deadline miss) is indeed a worthy goal. However, the technique that we propose lies in a different track: the adaptive reservation approaches. A basic ingredient of this class of algorithms is a resource reservation scheduler. Resource reservations have been introduced in [Mercer et al. 1993; Rajkumar et al. 1998] and they arguably offer a better support to multimedia applications than standard real-time scheduler. Indeed, they enable one to control the fraction of CPU (bandwidth) devoted to each application regardless of the workload generated by the other ones. The idea of the adaptive reservations is to use the bandwidth as an actuator to control the evolution of the tasks and has been first proposed in [Abeni and Buttazzo 1999]. As discussed in [Abeni et al. 2004], using the resource reservations algorithm allows us to model the evolution of the scheduling error of a single task by a discrete-time model and, hence, provide stability guarantees on the closed-loop system. In this paper, we take a step further in this direction, showing an adaptive reservation

algorithm and an analysis that allows us to make provisions on the probability of respecting the deadlines. Such guarantees are not given on one task in isolation, but the possible interaction with other tasks is explicitly considered.

The idea of adaptive reservation bears some evident resemblance with the notion of real-rate scheduling, proposed in [Steere et al. 1999; Goel et al. 2004], where a controller is used to regulate the progress rate of each task. The progress rate is defined as the difference between a time-stamp associated with a computation and the actual time this computation is performed. The main difference with adaptive reservations (and consequently with our approach) is that the latter adheres to the classical real-time tasking model, whereby an application is structured as a sequence of jobs and the control goals are referred to the deadlines of the jobs.

Another important research line tightly related to the approach presented in this paper is that of real-time algorithms that reclaim unused bandwidth [Lipari and Baruah 2000; Caccamo et al. 2005; Lin and Brandt 2005]. The general idea is that a task can use, for some job, less than its assigned budget. In such a case the spare budget can be reassigned to other active tasks. The differences between these algorithms lie in the rules for deciding when the unused budget can be reclaimed and which task takes benefit from the additional availability of budget. To this regard, Algorithms BASH [Caccamo et al. 2005] and BACKSLASH [Lin and Brandt 2005] reclaim all the remaining budget at the end of the task job and transfer it to the next task in the EDF queue. Therefore, such algorithms work well only with a strictly periodic task model. Algorithm GRUB [Lipari and Baruah 2000] does not make any assumption on the underlying task model: it instantaneously reclaims all extra bandwidth in the system and distributes it to the executing task. All these algorithms greedily give preference to the task with the earliest deadline, and it is not possible to favour one specific task over another. Using a technique similar to GRUB, in this paper we propose SHRUB, which has been designed to fairly distribute the spare bandwidth to active tasks according to user-defined weights.

More importantly, in this paper, we make the point that adaptive reservations and reclaiming algorithms are complementary techniques. Indeed, the former can be used in all conditions (including temporary overload), whereas the latter can only operate when the system is not saturated. Moreover, as discussed in the paper, the identification of correct bandwidth made by adaptive reservations improves the effectiveness of reclaiming even in nominal conditions. Finally, reclaiming unused bandwidth enhances the robustness of the feedback in achieving its goals.

9. CONCLUSIONS

In this paper, we propose a feedback scheduling algorithm, belonging to the family of adaptive reservations, which identifies and tracks the CPU requirements of multimedia tasks. The algorithm is combined with a global supervisor, which manages overload conditions and reclaims unused bandwidth using a fair algorithm called SHRUB. The combination of the two techniques is proved beneficial, since resource reclaiming increases the robustness of the feedback controller, while the feedback enables the CPU reclaiming algorithm to work properly. In order for the two mechanisms to co-exist nicely a "contract" has to be respected between the two components on the minimum bandwidth that the supervisor has to guarantee to

the task whenever the feedback controller requires it. We offer a theoretical analysis of the property than can be guaranteed whenever this contract is respected.

The whole mechanism has been implemented in the AQuoSA architecture and extensive experimental results have been collected both on real applications and on synthetic tasks that validate our approach.

REFERENCES

2004. ISO/IEC 14496-2:2004 - information technology – coding of audio-visual objects – part 2: Visual.

Abeni, L. and Buttazzo, G. 1998. Integrating multimedia applications in hard real-time systems. In *Proc. of the* 19^{th} *IEEE Real-Time Systems Symposium (RTSS 1998)*. IEEE, Madrid, Spain.

Abeni, L. and Buttazzo, G. 1999. Adaptive bandwidth reservation for multimedia computing. In *Proc. of the* 6^{th} *IEEE Real Time Computing Systems and Applications (RTCSA 1999)*. IEEE, Hong Kong.

Abeni, L. and Buttazzo, G. 2001. Hierarchical QoS management for time sensitive applications. In *Proc. of the* 7^{th} *IEEE Real-Time Technology and Applications Symposium (RTAS 2001)*. IEEE, Taipei, Taiwan.

Abeni, L., Cucinotta, T., Lipari, G., Marzario, L., and Palopoli, L. 2004. Adaptive reservations in a Linux based environment. In *Proc. of the* 10^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004)*. IEEE, IEEE, Toronto (Canada).

Abeni, L., Palopoli, L., Lipari, G., and Walpole, J. 2002. Analysis of a reservation-based feedback scheduler. In *Proc. of the* 23^{th} *IEEE Real-Time Systems Symposium (RTSS 2002)*. IEEE, Austin, Texas.

Baruah, S., Lipari, G., and Abeni, L. 2008. SHRUB: Shared reclamation of unused bandwidth. Tech. rep., Scuola Superiore Sant'Anna. July. http://retis.sssup.it/~lipari/papers/shrub_tech_report_jul_08.pdf.

Brandt, S. and Nutt, G. 2002. Flexible soft real-time processing in middleware. *Real-time systems journal, Special issue on Flexible scheduling in real-time systems 22,* 1-2 (January-March), 77–118.

C. Lu, J. Stankovic, G. T. and Son, S. 2002. Feedback control real-time scheduling: Framework, modeling and algorithms. *Real-Time Systems Journal, Special Issue on Control-Theoretic Approaches to Real-Time Computing 23,* 1/2 (September), 85–126.

Caccamo, M., Buttazzo, G. C., and Thomas, D. C. 2005. Efficient reclaiming in reservation-based real-time systems with variable execution times. *IEEE Transactions on Computers 54,* 2 (Feb.), 198–213.

Combaz, J. and Strus, L. 2008. A stochastic approach for fine grain QoS control. In *Proc. of the 2008 IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTImedia 2008)*. IEEE, Atlanta, GA, 115–120.

Corbato, F. J., Merwin-Dagget, M., and Daley, R. C. 1962. An experimental time-sharing system. In *Proc. of the AFIPS Joint Computer Conference*. ACM, Palo Alto, CA.

Cucinotta, T. 2008. Access control for adaptive reservations on multi-user systems. In *Proc. of the* 14^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2008)*. IEEE, IEEE, St. Louis, MO, United States.

Cucinotta, T., Palopoli, L., Marzario, L., and Lipari, G. 2008. AQuoSA – Adaptive Quality of Service Architecture. *Software – Practice and Experience 39,* 1, 1–31.

Faggioli, D., Checconi, F., Trimarchi, M., and Scordino, C. 2009. An EDF scheduling class for the Linux kernel. In *Proc. of the* 11^{th} *Real-Time Linux Workshop (RTLW 2009)*. OSADL, Dresden, Germany.

Falk, M. et al. 2006. A first course on time series analysis. http://statistik.mathematik.uni-wuerzburg.de/timeseries/.

Goel, A., Walpole, J., and Shor, M. 2004. Real-rate scheduling. In *Proc. of the* 10^{th} *IEEE Real-time and Embedded Technology and Applications Symposium (RTAS 2004)*. IEEE, Toronto (Canada), 434.

Hentschel, C., Bril, R., Gabrani, M., Steffens, L., van Zon, K., and van Loo, S. 2001. Scalable video algorithms and dynamic resource management for consumer terminals. In *Proc. of the International Conference on Media Futures (ICMF)*. AEI, Florence, Italy.

Hughes, C. J., Kaul, P., Adve, S. V., Jain, R., Park, C., and Srinivasan, J. 2001. Variability in the execution of multimedia applications and implications for architecture. *SIGARCH Comput. Archit. News 29,* 2, 254–265.

Isovic, D. and Fohler, G. 2004. Quality aware mpeg-2 stream adaptation in resource constrained systems. In *Proc. of the* 16^{th} *IEEE Euromicro Conference on Real-Time Systems*. IEEE, Catania, Italy.

Isović, D., Fohler, G., and Steffens, L. 2005. Real-time issues of mpeg-2 playout in resource constrained systems. *J. Embedded Comput. 1,* 2, 239–256.

Kleinrock, L. and Gail, R. 1976. *Queueing systems*. Wiley Interscience, New York.

Lan, T., Chen, Y., and Zhong, Z. 2001. Mpeg2 decoding complexity regulation for a media processor. In *Fourth IEEE Workshop on Multimedia Signal Processing*. IEEE, Cannes, France.

Lin, C. and Brandt, S. A. 2005. Improving soft real-time performance through better slack reclaiming. In *Proc. of the* 26^{th} *IEEE International Real-Time Systems Symposium (RTSS 2005)*. IEEE Computer Society, Washington, DC, USA, 410–421.

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

Liu, C. L. and Layland, J. 1973. Scheduling alghorithms for multiprogramming in a hard real-time environment. *Journal of the ACM 20,* 1, 46–61.

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

Nakajima, T. 1998. Resource reservation for adaptive QoS mapping in real-time mach. In *Proc. of the Sixth International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*. Springer Berlin / Heidelberg, Orlando, FL.

Rajkumar, R., Juvva, K., Molano, A., and Oikawa, S. 1998. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *Proc. of the SPIE/ACM Conference on Multimedia Computing and Networking*. SPIE, San Jose, California.

Roitzsch, M. and Pohlack, M. 2006. Principles for the prediction of video decoding times applied to mpeg-1/2 and mpeg-4 part 2 video. In *Proc. of the* 27^{th} *IEEE Real-Time Systems Symposium (RTSS 2006)*. IEEE, Rio de Janeiro, Brazil, 271–280.

Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V. 2003. Request for comments: 3550 – RTP: A transport protocol for real-time applications. `http://tools.ietf.org/html/rfc3550`.

Steere, D., Goel, A., Gruenberg, J., McNamee, D., Pu, C., and Walpole, J. 1999. A feedback-driven proportion allocator for real-rate scheduling. In *Proc. of the Third USENIX Symposium on Operating Systems Design and Implementation*. USENIX, New Orleans, LA.

Tokuda, H. and Kitayama, T. 1993. Dynamic QoS control based on real-time threads. In *Proc. of the* 4^{th} *International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 1993)*. Springer-Verlag, London, UK, 114–123.

Wüst, C. C., Steffens, L., Verhaegh, W. F. J., Bril, R. J., and Hentschel, C. 2005. QoS control strategies for high-quality video processing. *Real-Time Systems 30,* 1-2, 7–29.

Wüst, C. C. and Verhaegh, W. F. J. 2004. Quality control for scalable media processing applications. *J. Scheduling 7,* 2, 105–117.

THIS DOCUMENT IS THE ONLINE-ONLY APPENDIX TO:

A robust mechanism for adaptive scheduling of multimedia applications

TOMMASO CUCINOTTA*, LUCA ABENI†, LUIGI PALOPOLI†, GIUSEPPE LIPARI*
*Scuola Superiore Sant'Anna, †University of Trento

ACM Journal Name, Vol. V, No. N, Month 20YY, Pages 1–0??.

In this appendix, we report the proofs of the theorems introduced in the paper.

PROOF OF THEOREM 4.1. Given the evolution of the system in Equation (4), the event $\mathcal{H} = \{\mathcal{E}_{j+1} \leq 0\}$ can be expressed as $\mathcal{H} = \left\{\left\lceil \frac{\mathcal{C}_{j+1}}{Q_{j+1}} \right\rceil \leq N - S(\mathcal{E}_j)\right\}$. Observing that, for any integer n and for any real number x, we have $\lceil x \rceil \leq n$ if and only if $x \leq n$, the event $\mathcal{H}$ can be re-written as $\mathcal{H} = \left\{\frac{\mathcal{C}_{j+1}}{Q_{j+1}} \leq N - S(\mathcal{E}_j)\right\}$. Thereby, for any $\epsilon_j \leq R$:

$$\mathbf{Pr}\left\{\mathcal{H} \mid \mathcal{E}_j = \epsilon_j\right\} = \mathbf{Pr}\left\{\frac{\mathcal{C}_{j+1}}{Q_{j+1}} \leq N - S(\epsilon_j)\right\}$$

Assuming that $R \leq E_j$ (second theorem assumption), then the budget request given by Equation (8) $Q_{j+1} = H_{j+1}/(N - S(\epsilon_j))$ respects $Q_{j+1} \leq PU_{max}$, and it is granted by the supervisor due to the third theorem assumption. Hence,

$$\mathbf{Pr}\left\{\mathcal{H} \mid \mathcal{E}_j = \epsilon_j\right\} = \mathbf{Pr}\left\{\mathcal{C}_{j+1} \leq H_{j+1}\right\}.$$

The proof follows from the first theorem assumption in Equation (9). □

PROOF OF THEOREM 4.2. The proof is a direct consequence of the fact that, for any $\epsilon_j \leq R$, under the second theorem assumption, the control law in Equation (8), after the supervisor mediation, can not reach $\overline{Q}$ (see the saturation point M_j and the continuous line of Figure 4): $Q_{j+1} = \frac{H_{j+1}}{N - S(\epsilon_j)} \leq \frac{\sup_j H_j}{N-R} \leq \overline{Q}$. Therefore, the requested budget Q_{j+1} is always granted, and the same reasoning of Theorem 4.1 may be applied, with $\overline{Q}$ in place of PU_{max}. □

PROOF OF THEOREM 4.3. We start by proving the L bound. Assume that $\epsilon_{j_0-1} \leq M_{j_0-1}$, then applying Equation (4) with Q_{j_0} given by Equation (8), and exploiting $\lceil x \rceil \leq x + 1$, it is easy to get: $\epsilon_{j_0} \leq (\rho - 1)(N - S(\epsilon_{j_0-1})) + 1 \leq \overline{\epsilon}$, where $\overline{\epsilon} \triangleq (\rho - 1)N + 1$. If $\epsilon_{j_0} \leq M_{j_0}$, then we are already in a condition in which $\mathbf{Pr}\left\{\epsilon_{j_0+1} \leq 0\right\} \geq \pi$, so let us assume that $\epsilon_{j_0} > M_{j_0}$.

Now, we want to prove that, in a step $\tilde{j} \in \{j_0+1, \ldots, j_0+L\}$, it happens that $\epsilon_{\tilde{j}} \leq M_{\tilde{j}}$. To this purpose, focus on the path followed by the system in the subsequent V steps. If, for some $i \in \{1, \ldots, V\}$, we have that $\epsilon_{j_0+i} \leq M_{j_0+i}$, then we found $\tilde{j}$. In the opposite case, the budget required by the control law is always greater than $\overline{Q}$, and equal to $\overline{Q}$ in the worst-case. Furthermore, $\forall i \in \{1, \ldots, V\}$, $S(\epsilon_{j_0+i}) = \epsilon_{j_0+i}$. Therefore:

$$\begin{aligned}\epsilon_{j_0+V} &= \epsilon_{j_0} + \textstyle\sum_{i=1}^{V} \left\lceil \frac{c_{j_0+i}}{Q_{j_0+i}} \right\rceil - VN \leq \epsilon_{j_0} + \sum_{i=1}^{V} \frac{c_{j_0+i}}{\overline{Q}} - V(N-1) \leq \\ &\leq \epsilon_{j_0} + \frac{\sup_h \sum_{i=1}^{V} c_{h+i}}{\overline{Q}} - V(N-1) = \epsilon_{j_0} + \frac{V\phi_V}{\overline{Q}} - V(N-1) = \\ &= \epsilon_{j_0} - \theta\end{aligned}$$

where $\theta \triangleq V\left(N - 1 - \frac{\phi_V}{\overline{Q}}\right)$ is a strictly positive quantity due to the assumption in Equation (12). In other words, in a horizon of V steps, the scheduling error is reduced by at least $\theta > 0$. We can iterate the same reasoning until the condition $\epsilon_{j_0+\tilde{j}} \leq M_{j_0+\tilde{j}}$ is fulfilled, and the required number of V-steps iterations is bounded by the number of times θ needs to cover the distance between $\overline{\epsilon}$ and R (due to the theorem assumption about R). Therefore, we will reduce the scheduling error to below R, in at most $V\frac{\overline{\epsilon} - R}{\theta}$ steps, which corresponds to the expression for L in the theorem statement.

Now focus on the upper bound of the scheduling error evolution. Looking at the upper bound for ϵ_{j_0+V} above, as resulting in the second row, it is clear that the maximum of ϵ_{j_0+h} for $h \in \{1, \ldots, V\}$ may be expressed as follows:

$$\epsilon_{j_0} + \max_{h \in 1, \ldots V} \left\{ \sum_{i=1}^{h} \frac{c_{j_0+i}}{\overline{Q}} - h(N-1) \right\}.$$

An upper bound to this value can be found considering the computation times as free decision variables of an optimisation problem constrained by $\sum_{i=1}^{V} c_{j_0+i} \leq V\phi_V$. It can be easily seen that this maximum is attained for $c_{j_0+1} = V\phi_V$, and $c_{j_0+i} = 0$ for $i \in \{2, \ldots, V\}$, which causes the bound ϵ^{MAX} contained in the theorem statement to be attained for ϵ_{j_0+1}. The proof is obtained observing that, from ϵ_{j_0+V} on, the scheduling error can only be lower than such bound, due to the reduction of at least θ described above. □

Proof of Theorem 4.4. Assume that the conditions of Theorems 4.2 and 4.3 are met. We know that, if at job j_0 we have $\epsilon_{j_0} \leq M_{j_0}$, then: 1) $\epsilon_{j_0+1} \leq 0$ with probability π, 2) $\epsilon_h \leq M_h$ for some $h \in [j_0+1, j_0+L]$. Therefore, we can construct a Markov Chain (MC) like shown in Figure 11, with a state for each possible number of consequent states with $\epsilon_j > M_j$. Formally, the state S_0 represents the event $\epsilon_{j_0+1} \leq M_{j_0+1}$, state S_i represents the event that $\epsilon_j > M_j$ for i steps: $(\epsilon_{j_0+1} > M_{j_0+1}) \wedge \ldots \wedge (\epsilon_{j_0+i} > M_{j_0+i}) \wedge (\epsilon_{j_0+i+1} \leq M_{j_0+i+1})$. Due to the Theorem 4.3, we know such a MC has at most $L+1$ states. The transition probabilities p_i for this MC could be theoretically computed starting from the distributions of $\mathcal{C}_{j_0+1}, \ldots, \mathcal{C}_{j_0+L}$ and of $\mathcal{E}_{j_0}$ (note that one could numerically compute the latter distribution by exploiting a full MC with a state for each individual ϵ_j value – this

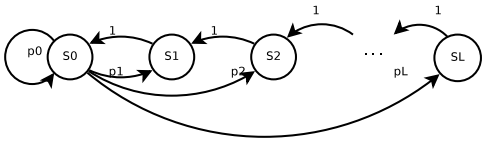

Fig. 11. Simplified Markov Chain

reasoning is only needed to show that the approach based on the MC in the figure is well-founded). However, we seek for closed-form bounds for $\mathbf{Pr}\{\epsilon_j \leq M_j\}$, what is achieved by considering upper and lower bounds for the MC transition probabilities p_i. In view of Theorem 4.2, we have $\pi \leq p_0 \leq 1$. Although the probabilities p_i are generally time varying, an upper bound for the probability of the state S_0 can be found $p_0 = 1$ and $p_i = 0$ for $i = 1, \ldots, L$, i.e., one never exits S_0. Likewise, a lower bound can be found setting $p_0 = \pi$, $p_1 = p_2 = \ldots = p_{L-1} = 0$ and $p_L = 1 - \pi$. In the latter case it is easy to find:

$$\mathbf{Pr}\{S_0\} = \mathbf{Pr}\{\epsilon_{j_0} \leq M_{j_0}\} \geq \frac{1}{1 + L(1-\pi)}.$$

The proof is completed observing that the state S_0 may actually be split into two substates, S_{00}, in which $\epsilon_{j_0} \leq 0$, and S_{01}, in which $0 < \epsilon_{j_0} \leq M_{j_0}$. In the search for the lower bound of $\mathbf{Pr}\{\epsilon_j \leq 0\}$, the arch from S_1 ends into S_{01}, the transition from S_{01} to S_{00} occurs with a probability of π, due to Theorem 4.2, S_{00} has an arch to itself with probability π, and transitions from both substates to S_L occur with a probability of $1 - \pi$. It is easy to verify that: $\mathbf{Pr}\{\epsilon_j \leq 0\} = \mathbf{Pr}\{S_{00}\} = \pi \mathbf{Pr}\{S_0\}$, corresponding to the expression in the theorem statement. □