Analysis of a Reservation-Based Feedback Scheduler *

Luca Abeni, Luigi Palopoli, Giuseppe Lipari,
Scuola Superiore Sant'Anna - Pisa
{luca, palopoli, lipari}@sssup.it

Jonathan Walpole
Oregon Graduate Institute, Portland
walpole@cse.ogi.edu

Abstract

When executing soft real-time tasks in a shared processor, it is important to properly allocate the computational resources such that the quality of service requirements of each task are satisfied. In this paper we propose Adaptive Reservations, based on applying a feedback scheme to a reservation based scheduler. After providing a precise mathematical model of the scheduler, we describe how this model can be used for synthesising the controller by applying results from control theory. Finally, we show the effectiveness of our method by simulation and by experiments with an MPEG player running on a modified Linux kernel.

1. Introduction

In recent years, a considerable amount of work has been done to apply real-time techniques to new kinds of time-sensitive applications, which are inherently more dynamic than classical real time applications.

Due to their temporal constraints, these systems are great candidates for using real-time techniques; however, they present new challenges due to the variability and unpredictability of their processing times, and to the data-dependent processing requirements that characterise them.

When multiple real-time tasks of this type share the same CPU, allocating resources to them becomes difficult, and the tasks can be properly served only if the resource scheduler is able to cope with the high variance and unpredictability of their requirements. Since the tasks running in the system are characterised by unpredictable behaviour, it is important to provide *temporal protection*, so that each task is protected from the fluctuations in the resource requirements of the other ones. When temporal protection is enforced, each task executes as if it were on a slower dedicated processor, and can be guaranteed independently of the presence of other tasks in the system. Temporal protection can be provided by using reservation-based schedulers [11], that have proven to be very effective in various different situations.

If the tasks' parameters were known in advance, it would be possible to statically reserve the proper amount of resources to each task. However, most of the time insufficient accurate information is available statically to reserve the correct amount of time to each task (because, for example, the execution times or the interarrival times are unknown a-priori).

It is clear that we need some way of dynamically adapting the scheduling parameters to the actual workload. We propose to do this, by using a feedback controller to monitor and adapt the reservation to the observed requirements. In other words, we believe that a combination of feedback scheduling techniques and resource reservations is a useful technique for properly serving time-sensitive applications in a modern multimedia OS. Some of the advantages of this combined use of feedback and reservation techniques are the following.

Better **portability** of real-time code: using a feedback reservation-based scheduler, the performance of the application does not depend on the execution time estimation. Thus, applications can easily run on lots of different machines and achieve a predictable QoS level.

A higher-level **programming interface**: the use of the proposed feedback mechanism permits the implementation of high level task models that separate the task parameters from the scheduling parameters [2].

Robustness to variations in execution times caused by DMA, caches, PCI bus masters, and similar mechanisms. Moreover, a reservation-based feedback scheduler permits to cope with the interrupt handling overhead by using augmented reservations [16] or similar mechanisms.

Increased system **efficiency**: adaptive reservations allow automatic adaptation of each reservation to the application's real requirements.

Using a reservation-based feedback scheduler, the problem of providing high system utilisation and high QoS to applications is decomposed into two simpler problems: 1) designing a feedback controller that is stable and provides

*This work was partially supported by DARPA/ITO under the Information Technology Expeditions, Ubiquitous Computing, Quorum, and PCES programs, NSF Grant CCR-9988440 and EIA-0130344, and by Intel.

a specified closed-loop dynamics, defined in terms of overshoot and response time, or in terms of closed-loop poles, and 2) choosing the closed loop dynamic that provides the desired QoS/utilisation trade-off.

In this paper, we address the first problem, characterising the open-loop behaviour of a reservation-based scheduler, and designing a proper feedback controller that is able to provide a specified closed-loop behaviour. We believe that the contribution of this paper is important in understanding the dynamics of a reservation-based scheduler, and providing a foundation for developing adaptive reservation techniques.

1.1. Related Work

Both the ideas of reservation-based scheduling and feedback scheduling are not new. CPU reservations were proposed by Mercer and others [11], and are the theoretical foundation for resource kernels [13]. Moreover, reservation techniques proved to be very effective in providing temporal isolation, and have been implemented in a number of different systems using different scheduling algorithms [5, 1, 4, 15]. Feedback techniques were originally proposed in time sharing systems [3], and have been successively applied to real-time [12, 16] and multimedia systems [18].

In particular, the reservation and feedback approaches can be combined to adjust tasks' periods according to the actual CPU load [12], or the CPU proportion of each process can be adapted to control the task's performance. For example, the processes served by reservation-based scheduler can be organised in a pipeline, and the the adaptation mechanism can control the length of the queues between pipeline's stages [18], or adaptive reservations can be used for separating the task parameters from the scheduling parameters [2].

Although these examples of feedback schedulers have proved to be very effective in addressing the above issues, little theoretical analysis of such mechanisms has been provided. For example, in some of those works [12, 2] the feedback scheduler is designed using an ad-hoc approach, whereas other works are based on the use of a PID controller [18], which is known to stabilise a wide range of systems. However, none of them analysed the feedback mechanism to guarantee its dynamic behaviour.

The use of a more theoretically founded approach (already presented in [7]) was finally advocated by Stankovic and others, and a feedback scheduler based on EDF was designed based on those premises [9]. However, when designing a feedback scheduler, control theory should be applied carefully. The authors realized the problem, and corrected their previous paper [8] presenting a general framework for evaluating feedback schedulers.

2. Terminology and Definitions

As discussed in the introduction, in this paper we propose to monitor the real-time performance of the tasks, and to use this information for adapting the amount of resources reserved to each task. This performance monitoring can be done by using a particular task model, the *real-time task model*, that permits the association of temporal constraints, called *deadlines*, with the task. If these temporal constraints are violated, the size of the reservation should be increased.

2.1. Definitions

According to the real-time task model, a task τ_i is a stream 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 $c_{i,j}$. Moreover, $J_{i,j}$ is characterised by a deadline $d_{i,j}$, that is respected if $f_{i,j} \leq d_{i,j}$, and is missed if $f_{i,j} > d_{i,j}$.

For the sake of simplicity, we will only consider *periodic tasks*, in which $r_{i,j+1} = r_{i,j} + T_i$, where T_i is the *task period*. Moreover, we will assume that $d_{i,j} = r_{i,j} + T_i$; hence, $r_{i,j+1} = d_{i,j}$.

Our goal is to provide support for time-sensitive applications in which a deadline miss can degrade the QoS of the task but does not have any catastrophic consequence. Therefore, we will consider *soft deadlines*, and the goal of the system will be to control the number of deadline misses.

2.2. Reservation-Based Scheduling

A reservation is a pair (B_i, T_i^s), where B_i is the fraction of resource utilisation dedicated to task τ_i, and T_i^s is the *period* of the reservation: task τ_i will be allowed to use the resource for $Q_i = B_i T_i^s$ units every period T_i^s. Q_i is also called the *budget* or *capacity* of the reservation.

It is important not to confuse the reservation period T_i^s with the task period T_i: although $T_i^s = T_i$ is a perfectly reasonable assignment, it is often useful to set the reservation period so that $T_i = kT_i^s, k \in \{1, 2, \ldots\}$. Although it is possible to define a reservation for all kind of resources (network, disk, etc.), in this paper we will consider only CPU reservations.

The behaviour of a reservation is the following: a task τ_i attached to a reservation (B_i, T_i^s) can execute for a time $Q_i = B_i T_i^s$ with a real-time priority, assigned according to some real-time scheduling policy, like Rate Monotonic or Earliest Deadline First. The reservation is often implemented using a *budget* q_i that is recharged to the maximum value Q_i at the beginning of every reservation period, and is decreased during the task execution. When the budget reaches 0, the reservation is said to be *depleted*: if the task needs to execute for more time, some action is required in order not to jeopardise the performance of the other tasks.

If $\sum_i B_i \le U^{lub}$, with U^{lub} depending on the scheduling algorithm upon which the reservation system is implemented, then each task τ_i attached to a reservation (B_i, T_i^s) is guaranteed to receive its reserved amount of time (i.e., Q_i time units over T_i^s).

For any reservation based system, we can define the *virtual finishing time* $\mathrm{VFT}_{i,j}$ of job $J_{i,j-1}$ as the time it would finish in a dedicated processor of speed B_i. For example, if job $J_{i,j-1}$ arrives at time $r_{i,j-1}$ and requests $c_{i,j-1}$ units of computation time, it would finish at time $r_{i,j-1} + \frac{c_{i,j-1}}{B_i}$ in the dedicated slower processor. Therefore, its virtual finishing time is $\mathrm{VFT}_{i,j} = \frac{c_{i,j-1}}{B_i}$. Intuitively, if $\mathrm{VFT}_{i,j} > T_i$, then we need to allocate a larger reservation to task τ_i, (i.e. we need to speed-up the dedicated processor) in order to fulfil its requirements.

It is also useful to define the concept of *latest possible finishing time* for a job. The latest possible finishing time $\mathrm{LFT}_{i,j}$ for job $J_{i,j-1}$ is the **end of the latest reservation period** used by the job, minus the job arrival time: for example, if $J_{i,j-1}$ has execution time $c_{i,j-1} = 5$, it has been reserved a bandwidth $B_i = 0.5$, and the reservation period is $T_i^s = 4$, then it uses $\lceil \frac{5}{0.5 \cdot 4} \rceil = 3$ reservation periods, and its LFT is 12 (whereas its VFT is $5/0.5 = 10$).

There is a clear relationship between VFT and LFT:

$$\mathrm{LFT}_{i,j} = \left\lceil \frac{\mathrm{VFT}_{i,j}}{T^s} \right\rceil T^s, \mathrm{VFT}_{i,j} = \mathrm{LFT}_{i,j} - \frac{q_i}{B_i} \qquad (1)$$

where q_i is the residual budget when the job finishes.

2.3. Bandwidth Reservations (CBS & GRUB)

Although any scheduling algorithm could be used to implement a reservation strategy, the use of a dynamic priority scheduler permits to obtain a more efficient implementation. An example is the Constant Bandwidth Server (CBS) [1] a reservation algorithm based on the Earliest Deadline First (EDF) priority assignment, that uses a server mechanism to implement reservations. A server is a *scheduling entity* that maintains two internal variables, a budget q_i and an absolute deadline d_i^s. Each time a task arrives, the server becomes active, and it is assigned an initial deadline d_i^s, and an initial budget q_i. Then, all the active servers are ordered in a EDF queue: the earliest deadline server is selected, and the corresponding task is executed. While the task executes, the server budget is decreased accordingly; if the task finishes before the reservation is depleted, the server becomes inactive and is extracted from the EDF queue. If instead the reservation is depleted and the task has not yet finished, the server deadline is postponed to $d_i^s = d_i^s + T_i^s$, and the EDF queue is reordered. For a full explanation of the algorithm, see [1].

Algorithm GRUB (Greedy Reclamation of Unused Bandwidth) [4] is very similar to the CBS. As with CBS, each server has an absolute deadline d_i^s; however, instead of using a budget q_i, each server uses a variable V_i, called *server virtual time*. In addition, each server uses an internal variable b_i that represents the current server bandwidth: if b_i is set constant and equal to B_i, then Algorithm GRUB is almost equivalent to Algorithm CBS[1]. If we want to reclaim the unused processor bandwidth, then b_i is variable and equal to $\frac{B_i}{B_t}$, where B_t is the sum of the bandwidths of the active servers. For the sake of simplicity, in the remaining of this paper we do not consider the reclaiming rule of Algorithm GRUB: hence, we set $b_i = B_i$.

Note that, for CBS and GRUB, when a job finishes the deadline of the server minus the job arrival time is equal to the latest possible finishing time: $\mathrm{LFT}_{i,j} = d_i^s - r_{i,j-1}$. Moreover, when using Algorithm GRUB, the VFT of a job is simply the value of the server virtual time when the job finishes.

2.4. Feedback Scheduling

A feedback mechanism can be defined based on: an **observed value**, used as input to the feedback mechanism, an **actuator**, which permits to apply the feedback action to change the system behaviour, and a **feedback law**, used to compute the new value to apply to the actuator, based on the observed value. When such a feedback scheme is applied to a scheduler, the observed value can be some QoS metric: for example, the deadline miss ratio in some interval of time [8], or the response time, the jitter, and so on. Similarly, some scheduling parameter can be used as an actuator.

If the feedback strategy is applied to a reservation based scheduler, the actuator is the amount of CPU reserved to the task (hence, Q_i or B_i). Since Q_i (B_i) is not constant, we will indicate it as $Q_{i,j}$ $(B_{i,j})$. We call the resulting abstraction an *Adaptive Reservation*. An adaptive reservation mechanism works as follows: a reservation $(B_{i,j}, T_i^s)$ is used to schedule τ_i; when a job $J_{i,j}$ finishes, an observed value is measured, and a new scheduling parameter $B_{i,j+1} = g(B_{i,j}, \ldots)$ is computed. If $\sum_i B_{i,j} > U^{lub}$, then the reserved bandwidths are rescaled, to maintain the system schedulable.

3. Mathematical Model of a Reservation

Since a proper feedback scheme providing the required characteristics can be designed only based on an accurate model of the system, we are going to develop a precise

[1]There is some slight difference in the initial assignment of the server deadline. However, this difference does not have a big impact on the dynamic behaviour of Algorithm GRUB.

mathematical model of a reservation based scheduler. First of all, we simplify the notation by removing the task index from all the quantities: we will use Q instead of Q_i, T^s instead of T_i^s, J_j instead of $J_{i,j}$, and so on.

The goal of our feedback scheduler is to control LFT (or VFT) to T; thus, we define the *scheduling error* ϵ_k as the difference between the latest possible finishing time LFT_k and the job relative deadline T. Note that, if $\mathrm{LFT}_k > T$, then job J_{k-1} consumes some of the time that should be used by the next job, which will have less time to execute. In this case, jobs J_{j-1} and J_j share a reservation period, and LFT_{j+1} depends on LFT_j. To express this dependency, and to write the dynamic equations of our system, it is useful to introduce another state variable that represents the amount of time used by J_{j-1} in its last reservation that it shares with J_j. We propose two different models: one model is based on the *virtual finishing time*, and we assume that all the state variables are accessible. The other model is based on the *latest possible finishing time* and we assume that LFT is the only accessible state variable.

3.1. Accessible Internal State

In the first model, we define the *scheduling error* as the difference between the virtual finishing time and the task period: $\epsilon_k = \mathrm{VFT}_k - T$. Using Algorithm GRUB VFT can be directly measurd; if, on the other hand, another reservation-based algorithm (like the CBS) is used and the value of the budget q is not accessible, then the VFT can be computed by using Equation 1.

If the scheduling error at the previous instance is less than 0, the VFT can be easily calculated as: $\mathrm{VFT}_k = \frac{c_{k-1}}{B_{k-1}}$. However, if the scheduling error is greater than 0, the VFT at step k depends on the value of the previous VFT: $\mathrm{VFT}_k = \mathrm{VFT}_{k-1} - T + \frac{c_{k-1}}{B_{k-1}}$.

By substituting, we can express the dynamic equation of the system as follows:

$$\epsilon_{k+1} = \begin{cases} \epsilon_k + \frac{c_k}{B_k} - T & \epsilon_k \geq 0 \\ \frac{c_k}{B_k} - T & \epsilon_k < 0 \end{cases} \qquad (2)$$

Note that, by using this definition, the scheduling error is a continuous value.

3.2. Non-Accessible Internal State

When it is not possible to measure the *virtual finishing time*, we define the scheduling error as the difference between the *latest possible finishing time* and the task period: $\epsilon_k = \mathrm{LFT}_k - T$. Notice that, in this case, the scheduling error is a discrete variable and it is a multiple of T^s.

We find it useful to define a state variable x_k that represents the amount time consumed by job J_{k-1} on the latest reservation period, if shared with job J_k. To help clarify the meaning of x_k, an example is shown in Figure 1. In Figure 1.a, J_1 uses only 2 reservation periods and finishes before the end of its period: J_1 and J_2 do not share any reservation, and $x_2 = 0$. In Figure 1.b, J_1 uses 3 reservation periods: therefore, $x_2 = 1$. In the following, we assume that x_k is not measurable.

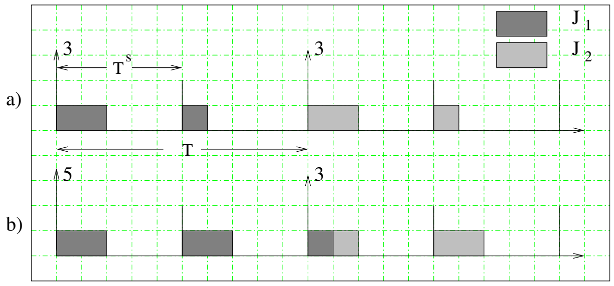

Figure 1. Internal state x_j.

The equations for computing x_k and LFT_k are shown in Figure 2; from those equations, we can derive the scheduling error:

$$\epsilon_k = \begin{cases} \epsilon_{k-1} - T - T^s + \left\lceil \frac{c_{k-1}+x_{k-1}}{B_{k-1}T^s} \right\rceil T^s & \epsilon_{k-1} \geq T^s \\ \left\lceil \frac{c_{k-1}}{B_{k-1}T^s} \right\rceil T^s - T & \epsilon_{k-1} < T^s \end{cases} \qquad (3)$$

Now, we want to show that this model is equivalent to the one expressed by Equation 2, plus a quantisation error due to the fact that the internal state x_k is not measurable.

We define the *quantisation error* QE_k in the two cases: $\epsilon_{k-1} \geq T_s$ and $\epsilon_{k-1} < T_s$. In the first case, ϵ_k depends on x_{k-1} that is not measurable. However, x_{k-1} is always in the range $[0, B_{k-1}T^s]$. Hence, we can use the following upper bound for the scheduling error:

$$\epsilon_k \leq \epsilon_{k-1} - T - T^s + \left\lceil \frac{c_{k-1} + B_{k-1}T^s}{B_{k-1}T^s} \right\rceil T^s$$

Now, we define the quantisation error as:

$$\mathrm{QE}_k = \left\lceil \frac{c_{k-1} + B_{k-1}T^s}{B_{k-1}T^s} \right\rceil - \frac{c_{k-1} + B_{k-1}T^s}{B_{k-1}T^s}$$

In the second case, we can simply define the quantisation error as:

$$\mathrm{QE}_k = \left\lceil \frac{c_{k-1}}{B_{k-1}T^s} \right\rceil - \frac{c_{k-1}}{B_{k-1}T^s}$$

Finally, we define a new scheduling error as $\tilde{\epsilon}_k = \epsilon_k - \mathrm{QE}_k T^s$. By substituting,

$$\tilde{\epsilon}_{k+1} = \begin{cases} \tilde{\epsilon}_k + \frac{c_k}{B_k} - T & \tilde{\epsilon}_k \geq T^s \\ \frac{c_k}{B_k} - T & \tilde{\epsilon}_k < T^s \end{cases} \qquad (4)$$

We can see Equations 2 and 4 are similar, except the switching point, which is 0 for Equation 2 and T^s for Equation 4.

$$\begin{cases} x_0 &= 0 \\ \mathrm{LFT}_1 &= \left\lceil \frac{c_0}{B_0 T^s} \right\rceil T^s \end{cases} \quad x_k = \begin{cases} c_{k-1} + x_{k-1} - (\mathrm{LFT}_k - T^s) B_{k-1} & \mathrm{LFT}_k > T \\ 0 & \mathrm{LFT}_k \le T \end{cases}$$

$$\mathrm{LFT}_k = \begin{cases} \left\lceil \frac{c_{k-1}}{B_{k-1} T^s} \right\rceil T^s & \mathrm{LFT}_{k-1} \le T \\ \mathrm{LFT}_{k-1} - T - T^s\left(1 - \left\lceil \frac{c_{k-1} + x_{k-1}}{B_{k-1} T^s} \right\rceil\right) & \text{otherwise} \end{cases}$$

Figure 2. Computation of x_k and LFT$_k$.

4. Controller Design

As shown in Section 3 a reservation-based scheduler with period T_s can be dealt with as a dynamical system described by the following equations:

$$\epsilon_{k+1} = \begin{cases} \epsilon_k + \frac{c_k}{B_k} - T & \text{if } \epsilon_k \ge K \\ \frac{c_k}{B_k} - T & \text{if } \epsilon_k < K \end{cases} \tag{5}$$

where ϵ_k represents the scheduling error, with $K = 0$ for the first model, described in Section 3.1, and $K = T^s$ for the second model, described in Section 3.2. In the latter case Equation 5 describes an approximation of the scheduling error where the quantisation error QE$_j$ is neglected (in the sequel we will also tackle this problem). The goal of this section is to propose techniques for effectively designing feedback controllers for this system.

We will first use a classical "pole-placement" technique to synthesise a controller in each mode. Before getting into these topics we need to introduce some definitions and notations that will be used throughout the section. By $\mathcal{C}_H$ we denote the set of sequences of execution times c_k such that $c_k < H$. Vector x_k will denote the state of the closed loop system, inclusive of ϵ_k and of the controller's own states (for example the state of a Proportional Integral (PI) controller consists of the past values of its input that it needs to issue a new command). The symbol $B_r(x_0)$ denotes the set of all points having euclidean distance (henceforth denoted by $dist$) from x_0 lower than or equal to r. Finally by $\overline{x}$ we will denote an equilibrium point for the closed loop system's state. An equilibrium $\overline{x}$ of the closed loop feedback scheduler is said *practically stable* if for all $\mu > 0$ and for all sequences $c_k \in \mathcal{C}_H$, there exists δ and R s.t. if $x_0 \in B_\delta(x_0)$ then $dist(x_k, B_R(\overline{x})) \le \mu$ for all k. Stronger concepts are practical *asymptotical* stability, for which $dist(x_k, B_R(\overline{x}))$ is required to vanish, and practical *exponential* stability that requires an exponential decay rate $\rho < 1$: i.e. $dist(x_k, B_R(\overline{x})) \le M\rho^k$.

It is worth noting that the action of the unknown disturbance c_k prevents one from controlling the system exactly into a point $\overline{x}$. Rather the system is controlled into a set $B_R(\overline{x})$, whose radius R grows with H (i.e. with the maximum allowed variation of the computation time). Consequently, ϵ_k that is part of the state is controlled into a neighbourhood $B_R(\overline{\epsilon})$ of the equilibrium $\overline{\epsilon}$ of radius R, given by a one dimensional projection of $B_R(\overline{x})$. Practical asymptotical stability implies that in response to a small perturbation of the initial state, the evolution of ϵ_k always remains close to $B_R(\overline{\epsilon})$ and that it is eventually captured into this set. For exponential stability, we include the additional requirement that the distance between ϵ_k and $B_R(\overline{\epsilon})$ decays with exponential rate. Another quantity of interest describing the quality of the system's evolution if the *overshoot* defined as $\max_k dist(x_k, B_r(\overline{x}))$.

4.1. Design based on a PI Controller

To design a controller for the dynamic system described by Equation 5, we are going to analyse the two modes corresponding to $\epsilon_k \ge K$ and $\epsilon_k < K$ separately. The underlying assumption is that that the considered equilibrium $\overline{\epsilon}$ is "far" from the switching surface $\epsilon_k = K$. If this assumption were released, there would be no theoretical support for the proposed design technique[2]. However, in practical applications the system did not exhibit pathological behaviour.

We are going to show the design for the first operating mode (the same considerations apply to the second one): if $\epsilon_k \ge K$, then $\epsilon_{k+1} = \epsilon_k + c_k u_k - T$ where u_k is defined as $\frac{1}{B_k}$.

Quantities ϵ_k, c_k, and B_k can be expressed as a constant value plus a variation: $\epsilon_k = \Delta\epsilon_k + \overline{\epsilon}$, $c_k = \overline{c} + \Delta c_k$ and $u_k = \overline{u} + \Delta u_k$. At the steady state it must hold $\overline{c} = \frac{T}{\overline{u}}$.

Assuming small variations around the linearization point, the relation between the variations can be found *via* differentiation:

$$\Delta\epsilon_{k+1} = \Delta\epsilon_k + \overline{c}\ \Delta u_k + \overline{u}\ \Delta c_k = \Delta\epsilon_k + \frac{T}{\overline{u}}\ \Delta u_k + \overline{u}\ \Delta c_k. \tag{6}$$

For notational simplicity, in the rest of the section we will drop the symbol Δ and, ϵ_k, u_k and c_k will represent variations of the original quantities around the $\overline{\epsilon}, \overline{u}, \overline{c}$ respectively.

[2] In the control literature it is possible to find both unstable systems resulting from the switching combination of stable systems and vice versa.

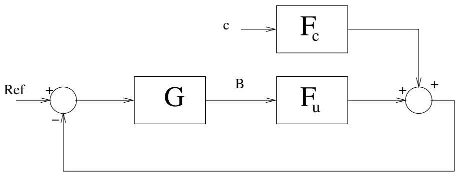

Figure 3. Dynamic system representing a linearised reservation with a feedback mechanism.

As the difference Equation 6 is linear, it is possible to compute the Z transform: $\epsilon(z) = F_c(z)c(z) + F_u(z)u(z)$, where $F_c(z) = \frac{\overline{u}}{z-1}$ and $F_u(z) = \frac{T}{\overline{u}(z-1)}$. To achieve the control goals, we use a feedback controller $G(z)$ as in Figure 3: $u(z) = -G(z)\epsilon(z)$. The closed loop dynamics is described by the transfer function $F(z)$ between $c(z)$ and $\epsilon(z)$:

$$\epsilon(z) = F(z)c(z) = \frac{F_c(z)}{1 + G(z)F_u(z)}c(z) \tag{7}$$

The simplicity of the system (whose dynamic equations are similar to those of a tank) suggested the use of a PI controller. A PI controller is described by: $u_k = c_1(-\epsilon_k) + c_2\sum_{j=0}^{k-1}(-\epsilon_{k-1})$ where c_1 and c_2 are the coefficients of the proportional and integral actions respectively. By subtracting the expression for u_{k-1} from the expression for u_k the equation can be written as: $u_k = u_{k-1} + \alpha(-\epsilon_k) + \beta(-\epsilon_{k-1})$, where $\alpha = c_1$ and $\beta = c_2 - c_1$. The transfer function $G(z)$ is given by:

$$G(z) = \frac{\alpha z + \beta}{z - 1}.$$

Plugging $G(z)$ into Equation 7, we have:

$$\epsilon(z) = F(z)c(z) = \frac{\overline{u}(z-1)}{z^2 + (\frac{T}{\overline{u}}\alpha - 2)z + \beta\frac{T}{\overline{u}} + 1}c(z). \tag{8}$$

As long as the equilibrium of the system is "far" from the switching surface and that variations around are small, requiring practical asymptotical stability is achieved if the zeros Z_i of the denominator in Equation 8 (i.e. the poles of the closed loop system), have norm strictly lower than 1: $||Z_i|| < 1$. Observe that the use of the PI controller enables the choice of the two closed loop poles poles. As a matter of fact, to place the closed loop poles in Z_1 and Z_2 it is sufficient to impose:

$$z^2 + (\frac{T}{\overline{u}}\alpha - 2)z + \beta\frac{T}{\overline{u}} + 1 = z^2 - (Z_1 + Z_2)z + Z_1 Z_2.$$

Solving for α, β yields:

$$\alpha = \frac{\overline{u}(2 - (Z_1 + Z_2))}{T}, \beta = \frac{\overline{u}(Z_1 Z_2 - 1)}{T}.$$

Moreover, the decay rate ρ is given by the maximum norm of the poles.

Repeating the computations for $\epsilon_{k-1} < K$, we obtain:

$$\alpha = \frac{\overline{u}(1 - (Z_1 + Z_2))}{T}, \beta = \frac{\overline{u}(Z_1 Z_2)}{T}.$$

4.2. Accounting for the Quantisation Error

Focusing on the case of the unaccessible internal state, we have to deal with the problem of quantisation error.

According to Equation 4, $\epsilon_k = \tilde{\epsilon}_k + \mathrm{QE}_k T^s$. As we did before, we consider an equilibrium point where the quantisation error has a value $\tilde{\mathrm{QE}}_k$ and repeat the analysis considering the variation around the equilibrium $\mathrm{QE}_k = \tilde{\mathrm{QE}}_k T_s + \Delta\mathrm{QE}_k$, where T_s has been absorbed into $\Delta\mathrm{QE}_k$. Hence we have $0 \leq \Delta\mathrm{QE}_k \leq T_s$. Considering now the linearised system, we can treat $\Delta\mathrm{QE}_k$ as an additional norm-bounded disturbance . The transfer function from such a disturbance to ϵ_k is given by $\frac{1}{1+F_u(z)\ G(z)}$.

If the controller is able to stabilise the system into a point rather than into a set, it is possible to apply the *Steady State Worst Case Analysis* developed by Slaughter [17]: the worst case steady state quantisation error on ϵ is lower than or equal to $|\frac{1}{1+F_u(z)G(z)}|_{z=1}T^s$. Replacing F_u and G with the expressions provided above, it is possible to conclude that $|\frac{1}{1+F_u(z)G(z)}|_{z=1}T^s = 0$. Therefore, if it is possible to stabilise the system into a point then the steady state value for the effect of the quantisation error is 0. The effect of quantisation is, in this case, an overestimation of the bandwidth $\tilde{B}$ assigned to the task. In fact, imposing the equilibrium condition $\epsilon_{k+1} = \epsilon_k$ in equation 3 we obtain:

$$\left\lceil \frac{\overline{c}}{\tilde{B}T^s} \right\rceil T^s - T = 0.$$

Observing that $x \leq \lceil x \rceil < x + 1$, we obtain:

$$\frac{\overline{c}}{T} \leq \tilde{B} \leq \frac{\overline{c}}{T - T^s}.$$

As one would expect, diminishing T_s (and hence the quantisation grain) results into higher and higher precision for the control.

4.3. A global stability test.

Such properties as the system's practical stability are formally guaranteed, in the synthesis technique proposed above, only if the closed loop evolution of the system is confined to one of the two modes (i.e. either $\epsilon_k \leq K$ or $\epsilon_k > K$ for all k).

In a special but important case it is possible to provide a stronger result. For the sake of simplicity, we restrict only to the case of accessible internal state, which is described by

Equation 5. Moreover, assume that the sequence of computation times c_k is constant - $c_k = \overline{c} \forall k$ - and that a lower and an upper bound are known: $h \leq \overline{c} \leq H$. For practical purposes this assumption means that the computation times vary slowly with respect to the system closed loop dynamics. For simplicity consider only the case when the system has to be stabilised into $\overline{\epsilon} = 0$, $\overline{u} = \frac{T}{\overline{c}}$. In this case the scheduler is a piecewise affine (PWA) system [6]: i.e. there exists a partition of the state space into polyhedral cells ($\epsilon_k \leq K$ and $\epsilon_k > K$) and in each cell the system evolves with a linear dynamic. We use a different PI controller in each cell:

$$u_k = \begin{cases} u_{k-1} - \alpha_1 \epsilon_k - \beta_1 \epsilon_{k-1} & \text{if } \epsilon_k \leq K \\ u_{k-1} - \alpha_2 \epsilon_k - \beta_2 \epsilon_{k-1} & \text{if } \epsilon_k > K \end{cases} \quad (9)$$

Considering as state vector $x_k = [\epsilon_k, u_k - \frac{T}{\overline{c}}]^T$, the closed loop evolution is given by:

$$x_{k+1} = \begin{cases} A_1(\overline{c}) x_k & \text{if } \epsilon_k \leq K \\ A_2(\overline{c}) x_k & \text{if } \epsilon_k > K \end{cases}$$

where:

$$A_1(\overline{c}) = \begin{bmatrix} 1 - \alpha_1 \overline{c} & \overline{c} \\ -(\alpha_1 + \beta_1) & 1 \end{bmatrix} \quad (10)$$
$$A_2(\overline{c}) = \begin{bmatrix} -\alpha_2 \overline{c} & \overline{c} \\ -(\alpha_2 + \beta_2) & 1 \end{bmatrix}$$

The problem we want to tackle is to analyse the robust stability of a given design. More precisely we want to know if a choice of the gains $\alpha_1, \alpha_2, \beta_1, \beta_2$ accomplishes the goal of stabilising the system *robustly* with respect to $\overline{c}$: i.e. for *any* value of $\overline{c}$ in the interval $[h,\ H]$. A sufficient test for this is the following:

Theorem 1 *The origin of the state space of the PWA system in Equation 10 is robustly asymptotically stable for* $\overline{c} \in [h,\ H]$ *if there exist a positive definite matrix* P *such that:*

$$\begin{aligned} A_1^T(h)\, P\, A_1(h) - P &\prec 0 \\ A_2^T(h)\, P\, A_2(h) - P &\prec 0 \\ A_1^T(H)\, P\, A_1(H) - P &\prec 0 \\ A_2^T(H)\, P\, A_2(H) - P &\prec 0. \end{aligned} \quad (11)$$

The notation $Q \prec 0$ in the above denotes that Q is negative definite matrix. Inequalities in condition 11 are linear matrix inequalities (LMI); finding a feasible solution for a system of LMIs is a problem that can be solved in polynomial time using convex optimisation techniques.

The above result produces a "global" stability test, i.e. we are able to know if the origin of the state space is asymptotically stabilised starting from *any* initial state and with the possibilities of switching between the two modes. The price to be paid for this strong result is the assumption of constant (or at least slowly varying) c_k.

5. Experimental Results

The correctness of the controller design was verified through simulations and through some experiments with real a implementation on the Linux kernel. In this section, we are going to show some of these results.

5.1. Simulations

Evaluating the performance of a feedback scheduler is not trivial: schedulers that seem to work properly at a first glance [9] may result to be unstable when evaluated more systematically [8]. To properly evaluate our adaptive reservation mechanism, we considered the system response to a step and a ramp in the system load, since they have been proven to be a good test case [8]. In particular, we report the evolution of the scheduling error ϵ_j and of the reserved CPU bandwidth B_j.

We performed extensive simulations using a wide set of different parameters. For the sake of brevity we report only some meaningful experiments. In particular, in the following we consider a task τ with period $T = 40$ and execution time $c_j = 5$ if $j < 300$, $c_j = 15$ otherwise, and we report only the results obtained using the model presented in Section 3.2, in which the internal state is not fully accessible, and that is more difficult to control.

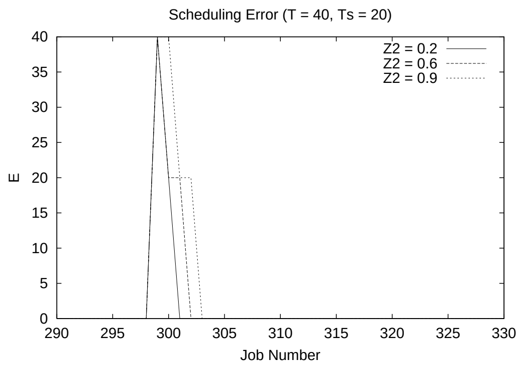

Figure 4. Scheduling Error obtained using an adaptive reservation with $T^s = 20$.

Figure 4 shows the evolution of the scheduling error for various values of the closed loop poles (in particular, $Z_1 = 0.1$, and $Z_2 = 0.2$, 0.6, or 0.9), when $T^s = 20$. When, at job J_{299}, the execution time increases from 5 to 15, the scheduling error raises to 40 (two times the reservation period), and it is controlled to 0 in a short time. Note that when the system reaches the steady state, the quantisation error is 0, as expected. Moving Z_2 from 0.2 to 0.9 the decay rate increases (as expected from control theory) thus resulting into a longer transient.

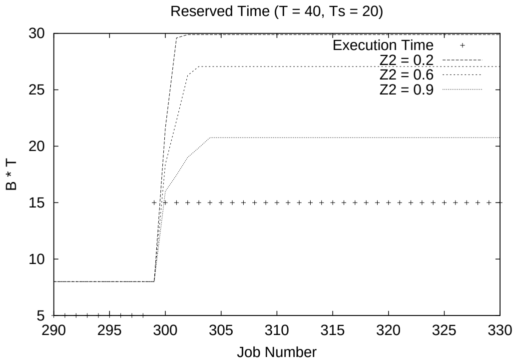

Figure 5. Bandwidth reserved by an adaptive reservation with $T^s = 20$.

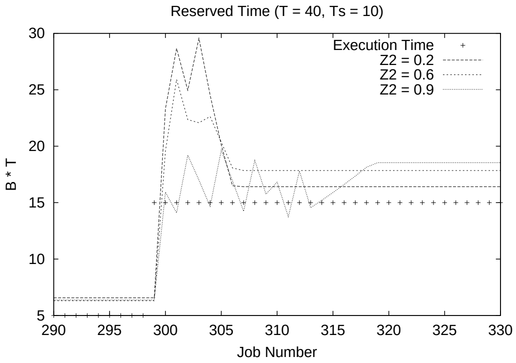

Figure 7. Bandwidth reserved by an adaptive reservation with $T^s = 10$.

Figure 5 shows the evolution of the reserved time, and is probably more interesting: the impact of the quantisation error is an overestimation of the reserved bandwidth, which in the worst case results to be 0.747198 instead of $0.375 = 15/40$. Hence, the overestimation is $0.747198 - 0.375 = 0.37220$; this value is compatible with the worst case estimation developed in Section 4.2, which is $B^0(T^s/(T-T^s)) = 0.375(20/(40-20)) = 0.375$. Note that, in this case, the quantisation error tends to increase when Z_2 moves to 0.2.

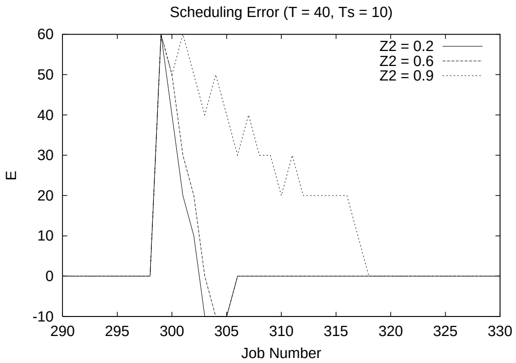

Figure 6. Scheduling Error obtained using an adaptive reservation with $T^s = 10$.

Figures 6 and 7 plot the evolution of the scheduling error and of $B_j T^s$ when $T^s = 10$, respectively. In this case, the quantisation error is lower and the response becomes closer to that of model without quantisation. In this case, faster controllers ($Z_2 = 0.2$ and $Z_2 = 0.5$) have an underrun in the scheduling error, that was previously masked by the quantisation error.

We repeated the same experiments using a ramp on the input, and we obtained similar results.

5.2. Real Workloads

As previously stated, the first set of experiments was performed based on a synthetic workload that has been recognised as particularly significant for evaluating system performance [8]. However, some experiments performed using a more realistic workload highlighted new problems.

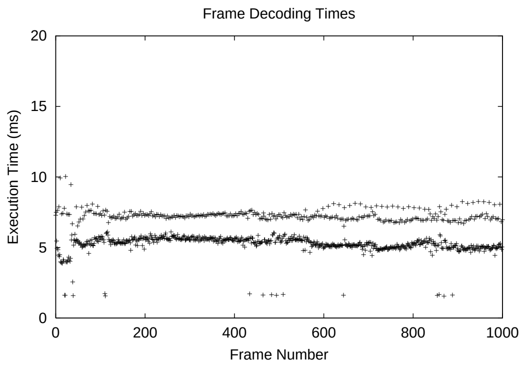

Figure 8. Frame decoding times for the Star Wars Episode 1 trailer.

To generate a realistic workload, we instrumented an MPEG player running on Linux, and we measured the frame decoding times for the trailer of Star Wars Episode 1 [10], shown in Figure 8 (note the huge variations in decoding times due to the VBR encoding). As it is possible to see, the execution times are highly variable. Since the goal of the PI controller is to control the scheduling error to 0, we can expect that this variability in the execution times will be reflected in a high variability in the reserved time. Figure 9

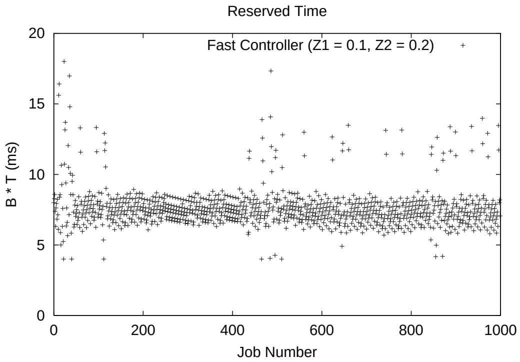

Figure 9. Reserved amount of time under a realistic workload (fast PI controller).

shows the evolution of the reserved time for a PI controller, using the second model (when the internal state is not fully accessible). We consider $T = 33ms$ (33.3 frames per second), $T^s = T/4 = 8.25ms$, $Z_1 = 0.1$ and $Z_2 = 0.2$ (the results obtained with the first model are similar). By comparing the two figures, it is clear that the reserved bandwidth does not stabilise properly; as a result, the scheduling error does not stabilise to 0, but continues to oscillate. We can expect this kind of problem, from the theory of control, because the system's input is highly variable. Since the system is practically stable and the variations in the input are bounded, the variations on the scheduling error are also bounded (and the average of the scheduling error is 0).

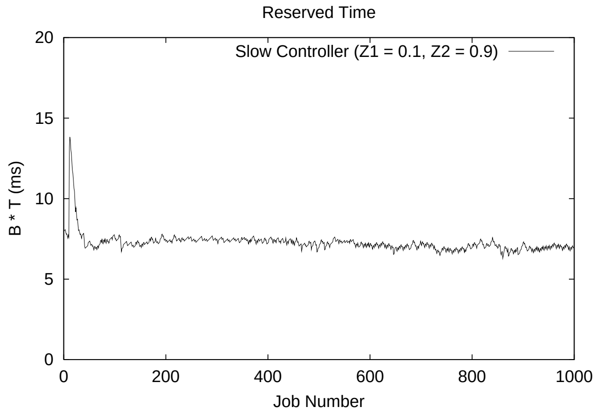

Figure 10. Reserved amount of time under a realistic workload (slow PI controller).

This oscillation of the scheduling error and of the reserved bandwidth can be addressed by filtering out the higher frequencies. We moved one of the two poles near to 1, and the results are shown in Figure 10. By comparing Figures 10 and 8, it is clear that the reserved bandwidth results to be more stable, and this can permit to better control the scheduling error. By analysing the scheduling error, it appears that the first controller (with $Z_2 = 0.2$) tends to "over-react" to execution time variations, presenting a bigger overshot: even after the initial transient, the scheduling error raises to more than $33ms$. On the contrary, moving the second pole to $Z_2 = 0.9$, the maximum scheduling error registered after the initial transient is $8.75ms$.

Summing up, we can say that while considering the response to a step or to a ramp the position of the poles Z_1 and Z_2 only influences the overshoot and the response time, when a more realistic workload is applied as input to the system, the position of the poles becomes critical for the system performance.

5.3. Implementation on a Real System

After verifying the correctness of the proposed feedback scheme through simulations (using synthetic and real-world workload traces), we performed some experiments on a real implementation. For this purpose, we used the dynamic QoS manager implemented on Linux/RK [14] and the second model of the scheduler, described in Section 3.2, which can introduce some quantisation error. The implementation of the PI controller presented in this paper was a simple task and required less than half an hour.

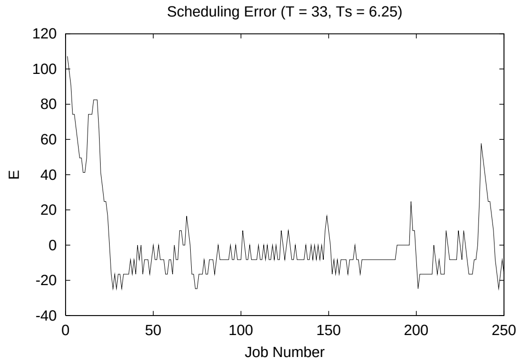

Figure 11. Scheduling Error for an MPEG player with $T = 33ms$ and $T^s = 6.75ms$.

Using this implementation, we tested the feedback scheduler by running two simultaneous MPEG players (at $33.3Fps$ and $20Fps$) attached to two adaptive reservations, with periods $33/4 = 8.25ms$ and $50/4 = 12.5ms$. The scheduling errors for the two players are shown in Figure 11 and 12. These experiments were performed setting $Z_1 = 0.1$ and $Z_2 = 0.8$.

After an initial transient, the feedback controller is able to adapt the reserved bandwidths so that the scheduling er-

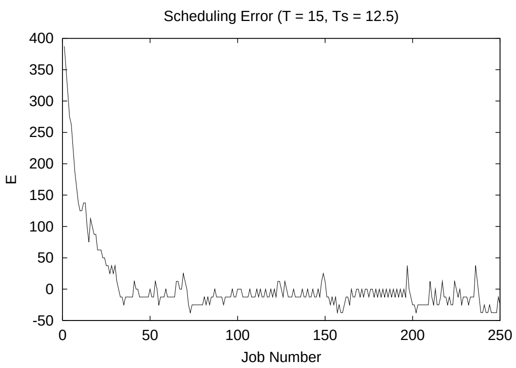

Figure 12. Scheduling Error for an MPEG player with $T = 50ms$ and $T^s = 12.5ms$.

ror is controlled to about 0. Since the execution times are highly variable, the scheduling error cannot be constant, but it is important to note that $\epsilon \leq 0$ most of the time (remember that a negative scheduling error is not bad for the perceived QoS). In coincidence with big variations in the execution times, the scheduling error increases, but it is immediately controlled to 0 again. It is important to note that these plots refer to *real experiments* performed on a *real Linux system*, and that the two players run simultaneously and share some important resource such as the X server.

6. Conclusions

In this paper, we addressed the problem of developing a feedback scheduler based on resource reservations, by developing a precise and accurate mathematical model of a reservation-based scheduler. We verified the correctness of the proposed model and the effectiveness of the designed controller through an extensive set of simulations and through real experiments on a real-time version of Linux. As a future work, we plan to apply control theory more rigorously in order to formally prove the robustness and stability of the controller. The problem of determining a proper closed loop dynamics that guarantees the desired QoS level in the presence of a realistic workload will be addressed. In particular, we will devise a methodology for imposing the decay rate of the close loop system.

References

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

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

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

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

[5] H. hua Chu and K. Nahrstedt. CPU service classes for multimedia applications. In *Proceedings of the IEEE International Conference on Mutimedia Computing and Systems*, Florence, Italy, June 1999.

[6] J. Johansonn and A. Rantzer. Computation of piecewise quadratic lyapunov functions for hybrid systems. *Transaction on Automatica Control*, 43(4), 1998.

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

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

[9] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Design and evaluation of a feedback control edf scheduling algorithm. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, Phoenix, AZ, December 1999.

[10] G. Lucas. Star wars episode I: The phantom menace, May 1999.

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

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

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

[14] R. R. Rajkumar, L. Abeni, D. de Niz, S. Ghosh, A. Miyoshi, and S. Saewong. Recent developments with linux/rk. In *Proceedings of the Second Real-Time Linux Workshop*, Orlando, Florida, november 2000.

[15] D. Reed and R. F. (eds.). Nemesis, the kernel – overview, May 1997.

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

[17] J. Slaughter. Quantization errors in digital control systems. *IEEE Transactions on Automatic Control*, 1964.

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