Efficient on-line schedulability test for feedback scheduling of soft real-time tasks under fixed-priority[*]

Rodrigo Santos,
Universidad Nacional del Sur
Bahía Blanca, Argentina.
ierms@criba.edu.ar

Giuseppe Lipari, **Enrico Bini**
Scuola Superiore Sant'Anna
Pisa, Italy
g.lipari@sssup.it,e.bini@sssup.it

Abstract

When dealing with soft real-time tasks with highly variable execution times in open systems, an approach that is becoming popular is to use feedback scheduling techniques to dynamically adapt the bandwidth reserved to each task. According to this model, each task is assigned an adaptive reservation, with a variable budget and a constant period. The response times of the jobs of the task are monitored and if different from expected (i.e. much larger or much shorter than the task relative deadline), a feedback control law adjusts the reservation budget accordingly. However, when the feedback law algorithm demands an increase of the reservation budget, the system must run a schedulability test to check if there is enough spare bandwidth to accommodate such increase. The schedulability test must be very efficient, as it may be performed at each budget update, i.e. potentially at each instance of a task.

In this paper, we tackle the problem of performing an efficient on-line schedulability test for Resource Reservation systems implemented through the Sporadic Server on Fixed Priority scheduling. We propose five different tests with different complexity and performance. In particular, we propose a novel on-line test, called Spare Pot algorithm *which shows a good cost/performance ratio.*

1 Introduction

A large class of embedded systems, including DVD and media players, TVs, teleconferencing systems, video servers, VoIP systems, etc. can be categorized as soft real-time. A soft real-time system is usually defined as a real-time system in which some of its deadlines can be missed without compromising the correctness of the results [8].

However, the number and severity of deadline violations may have a negative impact on the the Quality of Service (QoS) experienced by the user and on the amount of memory buffers that must be available. Using a large number of buffers implies a high end-to-end delay that may be unacceptable in applications like teleconferencing or live streaming. Also, a large amount of memory can increase the cost of the system.

Another important characteristic of such applications is the variability in resource requirements. In addition to well-known unpredictabilities of modern hardware architectures, designed for reducing average execution time, there are inherent variabilities due to the type of the application.

Finally, in this paper we address "open systems" in which new applications (processes or threads) can be activated dynamically to provide additional services, or to perform specific activities.

Scheduling In order to keep under control the application QoS under such a dynamic environment, it is important to use appropriate scheduling mechanisms. In this paper we propose to use the Resource Reservation Framework (RRF) [21]. In this framework, each task is associated a *reservation* characterized by a budget Q and a period P. Roughly speaking, a reservations *transforms* the soft real-time task into a sporadic task with worst-case execution time no larger than Q and minimum inter-arrival time equal to P, regardless of the actual task requirements. An admission control algorithm checks that the activation of a new reservation maintains the system schedulable. If the set of reservations is schedulable, the RRF provides the important property of *temporal isolation*: the ability of a task scheduled by a reservation to meet its timing constraints depends only on the reservation parameters and not on the presence of other tasks in the system. The task executes approximately as it were on a virtual dedicated processor of speed $U = Q/P$ times the speed of the real processor. Examples of such algorithms are the Constant Bandwidth Server running on top of Earliest Deadline First (CBS+EDF); the Sporadic Server

[*]This work has been supported in part by the EU Commission within the context of the FRESCOR EU project (FP6/2005/IST/5-034026).

running on top of Fixed Priority (SS+FP).

Budget assignment and feedback A correct budget assignment to reservations is critical for the good performance of a task. The problem can be solved by accurately profiling the computational requirements of the task. The budget can then be set to an appropriate value depending on the QoS requirements. However, in presence of soft real-time tasks with large variations of computational requirements, a static budget allocation may not be enough to guarantee a good performance.

A solution to this problem is represented by the *adaptive resource reservations* [2, 19]. In this approach, each reservation is assigned a feedback control module that measures the performance of the served task and tries to adjust the budget according to a certain control law. The *scheduling error* of a task [1] is defined as the difference between the actual finishing time of an instance of the task and its deadline. When the error is positive, it means the task has missed its deadline so the budget of its reservation was not sufficient and needs to be incremented. The control law tries to reduce the scheduling error to a value set by the user (usually equal to 0).

The budget of a reservation can be increased by the control algorithm only if there is enough spare bandwidth in the system to accommodate such increase. Suppose the feedback control modules dictates that the budget of a reservation must be increased by ΔQ. A schedulability test must be run to ensure that incrementing the budget, the other reservations remain schedulable. If this is not possible, the budget can be saturated to the maximum feasible value. Such test must be executed at each invocation of the control algorithm, i.e. at the end of each task instance. Therefore, the test must be as simple as it is possible, otherwise the overhead of checking feasibility would be so high that it will take a considerable portion of the processor bandwidth, nullifying the benefits of adaptive reservations.

1.1 Contributions of this paper

In this paper, the problem of efficiently managing the spare bandwidth in the system for adjusting the budgets of adaptive reservations via the feedback control is addressed. First, we propose a framework for implementing adaptive reservations regardless of the underlying scheduling algorithm. In particular, our framework will be valid for both the CBS algorithm on top of EDF, and for the Sporadic Server algorithm on top of FP. Then we concentrate on the SS+FP case. We propose five schedulability test with different degree of complexity and efficiency. Finally, we run comparative evaluation of all the five algorithms.

1.2 Related works

Many algorithms have been proposed in the past that implement the concept of adaptive reservations. A feedback law consisting in switching two traditional Proportional plus Integral (PI) controllers, based on a precise dynamical model of a resource reservation scheduler is introduced in [3] A control strategy that adjusts the bandwidth according to both the past execution time of the previous job and a prediction of the possible range of variation of the next execution time is given in [20]. New control approaches based on stochastic control are presented in [20]. The controllers are complemented by the use of a moving average filter to predict the execution time. In general, the prediction of the evolution of the execution times has shown to be very useful to improve the QoS of the task. However, the good response of the predictors used up to now depends on the specific application. All these results assume an EDF scheduler.

Many efforts have been made to apply traditional control techniques to optimize the performance of real-time schedulers. Some of the ideas focused on the sampling frequency of variables or task's periods, to cope with overruns on other tasks, are introduced in [9]. In [19], an analytical framework to map QoS requirements of adaptive real time systems into feedback control theory is proposed for any scheduling policy. However, the controller operates on admission thresholds and QoS discrete levels. The controller operates over the whole system and not in a per task/application way.

Concerning the problem of reclaiming spare bandwidth in resource reservation scheduling, many solutions have been proposed both in dynamic priority [17, 12, 11] and in fixed priority [5] reservation systems. However, it is important to point out that our focus here is completely different from the focus of the reclamation papers. In this paper we look for an efficient method for permanently increasing the budget of a reservation without affecting the other reservations. Reclamation algorithms try to greedily reclaim all spare bandwidth in the system at a certain time. However, this bandwidth is volatile, as it might not be available in the future, depending on the behavior of the other tasks. In our approach instead, we seek to permanently increase the budget of a reservation. Other differences will be presented in Section 2.1.

2 System model

We consider open systems where tasks may join or leave dynamically. A reservation $S_i = (Q_i, P_i)$ is thus characterized by a budget Q_i and a period P_i. The reservation bandwidth is denoted by $U_i = \frac{Q_i}{P_i}$. Reservations are implemented with appropriate aperiodic server algorithms. Many alternative implementations have been proposed, both in

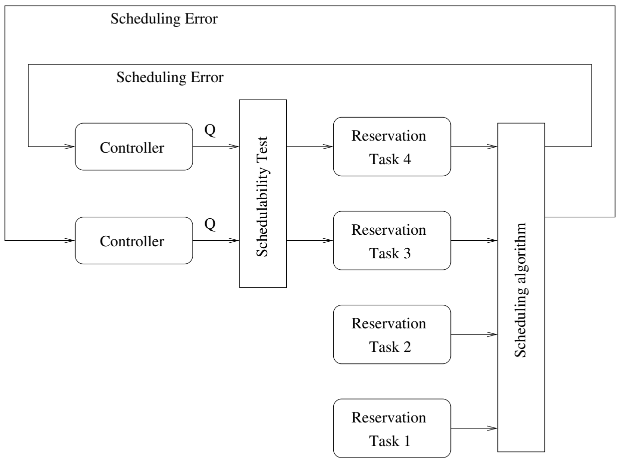

Figure 1. Architecture of the system.

static priority and dynamic priority servers. In this paper, we consider reservations implemented with the Sporadic Server [24] running on top of a FP scheduler (from now on, SS+FP), described in the real-time profile of the POSIX standard. Priorities are assigned according to the Rate Monotonic ordering between reservation periods.

The negotiation procedure guarantees that the allocated bandwidth does not exceed the schedulability bound. As a consequence, it consists in a schedulability test, where reservations are assimilated to sporadic tasks, with computation times equal to Q_i and period equal to P_i. Negotiation is a procedure that is seldom executed in the system, as the rate at which tasks ask to join or leave the system is very low. Also, the required deadline for a negotiation is often quite relaxed with respect to the typical rates in the system. For this reason, it is often implemented by an appropriate *service task* with its own reservation that can perform a complex schedulability test, like Response Time Analysis [4] or Sensitivity Analysis [7]. If the task is admitted and the new reservation created, it is guaranteed to execute Q_i unites of its budget within the reservation period P_i.

Reservations can be *fixed* or *adaptive*. In the first case, the budget Q_i never changes while the reservation is active. In the second case, instead, budget can be adapted quite frequently, even every reservation period, provided that there is enough free bandwidth in the system to accommodate for such a change. In particular, each reservation is associated a *feedback controller*, which monitors the performance of the task inside the reservation and measures its *scheduling error* [1]. The architecture of the system is shown in Figure 1. The scheduling error is a measure of how late (or of how early) a task completes with respect to its soft deadline [1]. Each request from a controller must be evaluated by a *supervisor* that performs a quick schedulability analysis to validate the request for change. If the request cannot be accommodated, it is saturated or rejected.

The feedback control law and the schedulability analysis performed by the supervisor must be extremely simple and the implementation must be efficient. In fact, in the worst-case, the feedback controller and the supervisor routine is invoked at each instance of a reservation. At the same time, they must be effective, as one of the requirements is to utilize the computational resource at its best.

2.1 Motivation for feedback scheduling

Consider a system consisting of 4 tasks, as shown in Figure 1. For the sake of simplicity, suppose that all reservation periods are equal. Thus, we can simply use Liu and Layland [18] utilization test with 1 as utilization bound: $\sum U_i \leq 1$. Two tasks require a constant bandwidth of 0.25, so they negotiate two reservations S_1 and S_2 with the same bandwidth. The other two tasks both require variable bandwidth. The minimum, average and maximum requirements are 0.1, 0.2 and 0.4, respectively. The situation is summarized in the table below (only the bandwidth requirements are reported for simplicity).

	U_{min}	U_{avg}	U_{max}
Task 1	0.25	0.25	0.25
Task 2	0.25	0.25	0.25
Task 3	0.1	0.2	0.4
Task 4	0.1	0.2	0.4
Total	0.7	0.9	1.3

Notice that, after the admission of the first two tasks, there is only 0.5 of spare bandwidth left, so we cannot successfully allocate 0.4 to both Task 3 and 4. Allocating bandwidth based on minimum values is not sufficient to accommodate for the task requests. Also, allocating a fixed bandwidth equal to the average values may not be a good solution: often, the reservation will not be able to satisfy the soft real-time task requirements, and the task deadline will probably be missed.

An alternative solution is to dynamically adapt the budget of the reservations corresponding to tasks 3 and 4, to take advantage of the statistical multiplexing property. If computation requirements of Task 3 and Task 4 are statistically independent, the probability that both require their maximum bandwidth at the same time is low. Therefore, the strategy consists in dynamically varying the budgets of reservations 3 and 4 so that each one gets the budget that needs when it most needs it. For example, it may be possible that when task 3 requires high bandwidth, task 4 requires a low bandwidth, so it is possible to accommodate for both.

Notice that there is an important difference between this approach and other dynamic reclamation mechanisms. Existing reclamation mechanisms (like the GRUB algorithm

[14], CASH [10], BASH [12]) try to reclaim spare capacity stemming from tasks that execute less than expected or by non-allocated bandwidth in the system. The reclaimed bandwidth is assigned in a greedy manner to the executing tasks, or divided among all active tasks according to some static weight.

The framework described in this paper, instead, dynamically assigns spare bandwidth to the task that most needs it *when* it needs it. In addition, feedback scheduling is rooted in control theory. As a consequence, under certain assumptions it is possible to analytically derive properties of the system (like stability, convergence, etc.).

3 The supervisor

An important component of the framework is the *supervisor*, that dynamically checks if the requests made by the feedback controllers can be accommodated. Basically, the supervisor must run a simple schedulability test which returns *yes* if a specific request for increasing the budget can be satisfied, or *no* if the request cannot be satisfied. In the second case, the test may additionally return the maximum allowed budget increase.

In this paper, we consider many different schedulability tests for fixed priority scheduling, at different levels of complexity. First, we consider tests based on utilization bounds [18, 13, 15]. Then we consider more complex tests based on the concept of Scheduling Points [16, 6]. Third, we consider the Response Time Analysis (RTA) [4], which provides the worst-case response time of each task in the system, and propose a new approximate schedulability test based on it specifically devised for the problem of feedback scheduling. The tests we consider here have different performance and different complexity. In Section 4, we compare them from a performance point of view and propose a trade-off between complexity and performance.

3.1 Scheduling Points algorithms

A schedulability test based on Scheduling Points consists in the following equation

Theorem 1 (from [16]) *A set of reservations* $\{(Q_i, P_i)\}$ *is schedulable under FP* ***if and only if***

$$\forall i = 1, \dots, n \; \exists t \in \mathsf{schedP}_i \quad Q_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{P_j} \right\rceil Q_j \leq t \quad (1)$$

where schedP_i *is the set of* scheduling points *[16, 6].*

By introducing the logical AND ($\wedge$) and the logical OR ($\vee$) operators and a more compact vectorial notation $\mathbf{U} = (U_1, \dots, U_n)$, the Eq. (1) can be rewritten as

$$\bigwedge_{i=1,\dots,n} \; \bigvee_{t \in \mathsf{schedP}_i} \boldsymbol{\alpha}(i,t) \cdot \mathbf{U} \leq 1 \quad (2)$$

where $\boldsymbol{\alpha}(i,t)$ is defined as

$$\boldsymbol{\alpha}(i,t) = \left(\underbrace{\left\lceil \frac{t}{P_1} \right\rceil \frac{P_1}{t}, \dots, \left\lceil \frac{t}{P_{i-1}} \right\rceil \frac{P_{i-1}}{t}}_{1,\dots,i-1}, \underbrace{\frac{P_i}{t}}_{i}, \underbrace{0, \dots, 0}_{i+1,\dots,n} \right)$$

The complexity of testing the schedulability by Eq. (2) is essentially due to the number of scheduling points in schedP_i, which in the worst-case is 2^{i-1} [6].

The advantage of using this test is that Eq. (2) can be immediately used to find the maximum admissible variation of any reservation bandwidth which does not compromise the feasibility. In fact, from the sensitivity analysis of FP systems [7] it follows that the amount of admissible variation ΔU_k to the bandwidth of the reservation S_k is

$$\Delta U_k^{\mathsf{exact}} = \min_{i=k,\dots,n} \; \max_{t \in \mathsf{schedP}_i} \frac{t - \boldsymbol{\alpha}(i,t) \cdot \mathbf{U}}{\alpha_k(i,t)} \quad (3)$$

where $\alpha_k(i,t)$ denotes the k^{th} component of the vector of coefficients $\boldsymbol{\alpha}(i,t)$. We refer to the evaluation of $\Delta U_k^{\mathsf{exact}}$ by means of the Eq. (3) as the exact method since it is derived from a necessary and sufficient condition.

Below we propose a simple example with only two reservations. Suppose we have $Q_1 = 2$, $P_1 = 5$ and $Q_2 = 1$, $P_2 = 8$. By applying the definition of the scheduling points [6], we find that $\mathsf{schedP}_1 = \{5\}$ and $\mathsf{schedP}_2 = \{5, 8\}$. Now it is possible to compute explicitly the inequalities resulting from Eq. (2) which are

$$U_1 + \frac{8}{5} U_2 \leq 1 \qquad \text{from } t = 5 \quad (4)$$

$$\frac{5}{4} U_1 + U_2 \leq 1 \qquad \text{from } t = 8 \quad (5)$$

and to represent them graphically (see Figure 2(a)). In the figure the black dot at the point $U_1 = \frac{2}{5}, U_2 = \frac{1}{8}$ denotes the initial utilization requirement, whereas the dark gray area denotes the region where the utilization will move into, during the run-time of the system. Eq. (3) allows to compute the maximum admissible variation to each utilization starting from the initial point. For the values of this example we have:

$$\Delta U_1^{\mathsf{exact}} = \frac{2}{5} = 0.4 \quad (6)$$

$$\Delta U_2^{\mathsf{exact}} = \frac{3}{8} = 0.375 \quad (7)$$

Although for the two tasks case the computation of the maximum variation is very simple, as the number of tasks

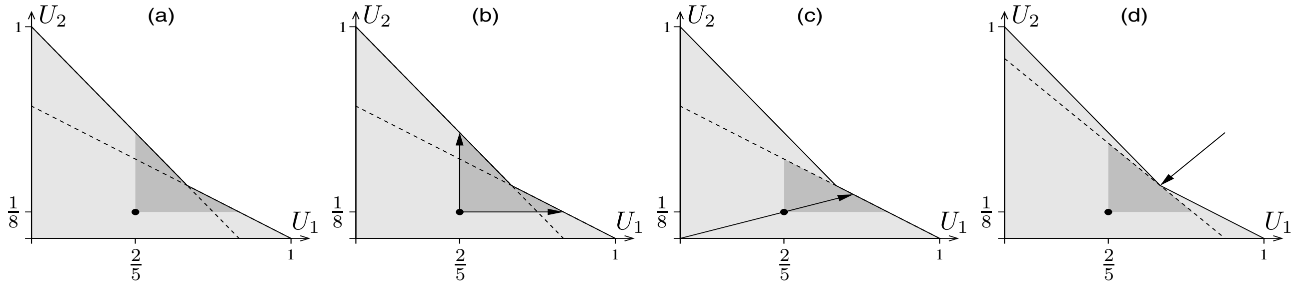

Figure 2. Scheduling Point based methods

grows the evaluation of Eq. (3) requires very long time. In fact, the complexity of the exact method is $O(n\,2^n)$ since it depends on the number of points in the set schedP_i. Hence the evaluation of the amount of ΔU_k at run-time is impractical. For this reason we investigate also other methods to trade accuracy for an efficient computation.

The simplification of the computation of the admissible variation is enabled by the following result.

Corollary 1 *Let $\{\mathsf{smallSet}_i\}_{i=1,\dots,n}$ be a family of subsets of schedP_i (meaning that $\forall i\ \mathsf{smallSet}_i \subseteq \mathsf{schedP}_i$), then the task set is schedulable **if**:*

$$\forall i = 1,\dots,n\ \exists t \in \mathsf{smallSet}_i\ \ Q_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{P_j} \right\rceil Q_j \le t \quad (8)$$

The proof is ommited here for space reasons but can be found in [22].

Corollary 1 suggests that as we find a smaller set of scheduling points, the amount of admissible variation of the utilization can be efficiently computed at the price of accuracy. The idea we will exploit next is to explore a neighborhood of the initial point so that only the constraints which are "close" to the starting point are considered.

The intersect method In this method we select a subset $\mathsf{intersectSet}_i$ of the scheduling points. To build the set $\mathsf{intersectSet}_i$, we analyze the maximum admissible variation of the utilization U_k for $k = 1,\dots,i$ so that the task τ_i is schedulable. For every pair (i,k) we add to the set $\mathsf{intersectSet}_i$ the scheduling point which originates an inequality which holds with the equal sign in Eq. (2). Following this procedure the number of points in $\mathsf{intersectSet}_i$ never exceeds i. Since we have to consider all tasks, then the total number of constraints is no more that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$. Therefore, the complexity of this test is $O(n^2)$.

The selection procedure is also depicted in Figure 2(b). Starting from the initial assigned values (the black dot), it is computed the maximum variation along both the directions and the hit constraints are stored in the set of reduced number of scheduling points.

The scaling method In this method, we select only one scheduling point for each task, which results in a total of n constraints to be checked at run-time. As shown in Figure 2(c) we select the constraint which is hit when we scale linearly all the utilizations.

In the proposed example the constraint corresponding to the scheduling point $t = 5$ is selected, so that $\mathsf{scalingSet}_2 = \{5\} \subset \mathsf{schedP}_2$. If we compute the maximum admissible variation by this method we find

$$\Delta U_1^{\mathsf{scaling}} = \frac{2}{5} = 0.4 \quad (9)$$

$$\Delta U_2^{\mathsf{scaling}} = \frac{1}{4} = 0.25 < \Delta U_2^{\mathsf{exact}} \quad (10)$$

The complexity of this method is linear $O(n)$.

The upBound method The final method we propose is not based on the scheduling points, but on the idea of utilization upper bound [18, 13]. We remind that the utilization upper bound for level-i tasks, is the maximum $U_{\mathsf{ub}}^{(i)}$ such that the condition

$$\sum_{j=1}^{i} U_j \le U_{\mathsf{ub}}^{(i)} \quad (11)$$

implies that the task τ_i is schedulable.

In the proposed example of two tasks, the utilization upper bound is $U_{\mathsf{ub}}^{(2)} = \frac{17}{20} = 0.85$ and the resulting maximum admissible variation of the utilizations are

$$\Delta U_1^{\mathsf{upBound}} = 0.325 \quad (12)$$

$$\Delta U_2^{\mathsf{upBound}} = 0.325 \quad (13)$$

This method has constant complexity $O(1)$.

3.2 RTA-based mechanism

In this section we present the Spare Pot algorithm for managing the spare bandwidth in SS+FP. It is an approximation of the RTA schedulability test for the purpose of checking the schedulability in the neighborhood of a schedulability working point.

The basic idea is the following. Suppose the system consists of a number n of adaptive reservations $S_1, S_2, \ldots, S_n$, in decreasing order of priority. Other non-adaptive reservations can be present in the system with arbitrary priority levels. The schedulability of such reservations is checked by using the RTA. Once again, each reservation is considered as a sporadic task with worst-case computation time Q and minimum inter-arrival time P. By using the RTA, we can compute the *worst-case response time* of each reservation (notice that this is only indirectly related to the response time of the task).

In addition, the user initially negotiates an additional *fake* reservation S_0 with higher priority than all adaptive reservations. This reservation is called *spare pot*: its only purpose is to reserve a bandwidth that can be collectively reclaimed by any of the adaptive reservations. Notice that this bandwidth is reserved: the on-line negotiation algorithm cannot reclaim it to make space for newly incoming reservations.

At run time, an adaptive reservation can ask the supervisor to decrease or increase its budget. In the first case, we say that the reservation *donates* part of its budget, while in the second case we say that the reservation *borrows* a part of its future budget.

A reservation at priority level i can only donate budget to lower priority reservations, and can only borrow from higher priority reservations. A data structure is maintained by the algorithm to keep track of how much budget each reservation S_i has donated to each other reservation S_j. The algorithm consists of a *preparation phase*, to be done at negotiation time, and of an *on-line phase*. The main steps of the algorithm are detailed in the following.

Preparation phase This phase is composed by the following steps.

§**1** Reserve a budget for the spare pot. This reservation will not directly execute any task; it is used mainly for preventing the admission control from allocating all spare bandwidth to newly incoming reservations. We denote this reservation with S_0, and its budget and period are Q_0 and P_0, respectively.

§**2** Perform RTA for admission control of new reservations. The equation to be checked is:

$$\forall i \min R_i | R_i = \sum_{j=0}^{i-1} \left\lceil \frac{R_i}{P_j} \right\rceil Q_j + Q_i \tag{14}$$

where (Q_j, P_j) are the parameters of the j-th reservation. All reservations are ordered by decreasing priority. R_i is the smallest solution of the fixed point equation. Assume all response times are less than or equal to the servers's relative deadlines $R_i \leq P_i$.

§**3** The algorithm stores the number of instances of adaptive reservation S_j that preempt reservation S_i in the worst case, in variable $\eta_{i,j}$:

$$\eta_{i,j} = \left\lceil \frac{R_i}{P_j} \right\rceil \tag{15}$$

In addition, it computes the following variable:

$$m_{i,j} = \min \left\{ \eta_{i,j}, \min_h \left(\frac{\eta_{h,j}}{\eta_{h,i}}\right) \right\} \tag{16}$$

§**4** Let n be the number of reservations currently in the system. The algorithm builds a square matrix $\Pi = (\pi_{i,j})$ of size $n+1$, with rows and columns numbered from 0 to n. Each element $\pi_{i,j}$ of the matrix has the following meaning:

- if $i < j$, element $\pi_{i,j}$ represents the units of budget that reservation S_i has donated to S_j (can only be non-negative).
- if $i = j$, element $\pi_{i,i}$ can only be negative or 0. Quantity $-\pi_{i,i}$ is the amount of budget that S_i has made available to others. The value of $\pi_{0,0}$ is constant and equal to $\pi_{0,0} = -Q_0$ (the budget of the Spare Pot reservation).
- if $i > j$, element $\pi_{i,j}$ represents the amount of budget that S_i has borrowed from S_j. Can only be non-negative.

Initially, all elements of the matrix are set to 0, except for $\pi_{0,0}$ that contains the negative of the Spare Pot budget. We define $\delta_i = -\sum_{j=0}^{n} \pi_{i,j}$ as the amount of borrowed budget (if negative) or the amount of budget still available (if positive) at priority level i. Notice that, for the spare pot S_0, the available budget at any instant is $\delta = -\sum_{j=0}^{n} \pi_{0,j}$.

On-line phase During execution, whenever a feedback controller requires a change in the budget of reservation S_i, the matrix is updated as follows.

- Suppose that the feedback controller needs to increase the budget of reservation S_i by $\Delta Q_i > 0$. The algorithm looks at δ_j, with $j = i, i-1, \ldots, 1, 0$. If $\delta_j \leq 0$, then it is not possible to borrow from this priority level, and we look at the next level $j-1$. If $\delta_j > 0$, then the maximum amount of budget it is possible to borrow from this level is $\delta_j m_{j,i}$. Let $x_j = \min\{\Delta Q_i, \delta_j m_{j,i}\}$. Then the matrix is updated as follows:

$$\begin{aligned}\pi_{i,j} &= \pi_{i,j} + x_j \\ \pi_{j,i} &= \pi_{j,i} + \frac{x_j}{m_{j,i}}\end{aligned}$$

If $x_j < \Delta Q_i$, then $\Delta Q_i = \Delta Q_i - x_j$ and we look at the next higher priority level. If we reach the last priority level $j = 0$ and $\Delta Q_i > 0$, then the remaining is discarded (*saturation*). The algorithm returns the cumulative sum of allocated budget.

- Suppose that the feedback controller requires to decrease the budget of reservation S_i by $\Delta Q_i < 0$. First, the algorithm tries to give back the budgets that S_i had borrowed from higher priority levels to their respective owners. Hence, select $j = 0, 1, \ldots, i-1$, and let $x_j = \min\{\pi_{i,j}, -\Delta Q_i\}$. Then the matrix is updated as follows:

$$\begin{aligned}\pi_{i,j} &= \pi_{i,j} - x_j \\ \pi_{j,i} &= \pi_{j,i} - \frac{x_j}{m_{j,i}}\end{aligned}$$

  Then $\Delta Q_i = \Delta Q_i + x_j$. If ΔQ_i is 0, then the algorithm stops. Otherwise, we continue with the next j. If after $j = i - 1$ we still have $\Delta Q_i < 0$, then we update $\pi_{i,i} = \Delta Q_i$, making the extra budget available to lower priority levels.

The following example shows how the algorithm works. Consider again the example of Section 3.1, consisting of two tasks τ_1 and τ_2, with $C_1 = 2$, $T_1 = 5$ and $C_2 = 1$, $T_2 = 8$. The spare pot consists in a reservation with $Q_s = 2$ and $P_s = 5$. The response times of the two reservations are $R_1 = 4$ and $R_2 = 5$, respectively. Finally, $m_{0,1} = m_{0,2} = m_{1,2} = 1$.

The initial value of the matrix are shown in Figure 3(a). Suppose that reservation S_1 releases 0.3 units of budget. The corresponding matrix after the update is shown in Figure 3(b). Now, suppose that reservation S_2 asks to increase its budget by 0.5. In Figure 3(c), the values after the increment are shown. Notice that in this particular example $\pi_{i,j} = \pi_{j,i}$. However, this is not always the case, as it depends on the fact that all $m_{i,j}$ are equal to 1.

3.2.1 Formal analysis of correctness

To demonstrate the correctness of the algorithm, it must be shown that the worst-case response times of all reservations do not change when changing the budget of any adaptive reservation. Let's start by discussing some properties of the matrix data structure.

Property 1 *Consider a reservation S_i. At any instant t, let $\overline{Q_i}$ be its current budget. Then, $\overline{Q_i} = Q_i + \sum_{j=0}^{i-1} \pi_{i,j} + \pi_{i,i}$. More specifically, if S_i borrowed budget from higher priority levels, then $\pi_{i,i} = 0$ and $\sum_{j=0}^{i-1} \pi_{i,j} \geq 0$. On the contrary, if S_i has made available some budget for lower priority levels, then $\sum_{j=0}^{i-1} \pi_{i,j} = 0$ and $\pi_{i,i} \leq 0$.*

Lemma 1 *For each row i of the matrix, one of the two following cases is true: either the first $i - 1$ elements are all 0; or if a positive element exist, then the following $i + 1$ elements must all be zero.*

Proof. Follows directly from Property 1. □

Property 2 *Since a reservation cannot lend more than it has made available, then $\sum_{j=i+1}^{n} \pi_{i,j} \leq -\pi_{i,i}$.*

For the sole purpose of demonstrating the theorem, the matrix is extended to include fixed adaptive reservations. A row is added for each fixed reservation at the appropriate position. The additional rows will have all elements equal to 0, and those elements will never change. Let n be the total number of reservations in the system (excluding the Spare Pot).

Theorem 2 *For any reservation S_i, let $\overline{Q_i}$ be its budget at time t, as modified by one or more iterations of the Spare Pot algorithm. If the reservation is fixed, then $\overline{Q_i} = Q_i$. Let $\overline{R_i}$ be the worst-case response time of reservation S_i computed considering each reservation S_j, with $j = 0, \ldots, i$ to have a budget $\overline{Q_j}$. Then $\overline{R_i} \leq R_i$, where R_i is the response time of reservation S_i, as computed during the preparation phase.*

The proof is ommited for space reasons. The interested reader can find it in [22].

3.2.2 Complexity of the method

In the first phase (during admission control) the complexity of this method is the same as of RTA, that is pseudo-polynomial ($O(n^2 T_{max})$). This can be very large, however, some techniques exists for reducing the average case behavior of RTA [23].

The on-line phase has complexity linear in the number of adaptive reservations ($O(n)$), as in the worst-case the lowest priority adaptive reservation has to check all δ_i of the higher priority reservations. However, the average-case behavior of the algorithm can be much lower for two reasons: first, the higher priority reservations have to check for less rows of the matrix in the worst-case; second, in the best case the reservation with priority immediately above the reservation under analysis may already be able to provide the needed

	S_0	S_1	S_2	δ_i	$Q_i(t)$
S_0	-2	0	0	2	2
S_1	0	0	0	0	2
S_2	0	0	0	0	1

(a)

	S_0	S_1	S_2	δ_i	$Q_i(t)$
S_0	-2	0	0	2	2
S_1	0	-0.3	0	0.3	1.7
S_2	0	0	0	0	1

(b)

	S_0	S_1	S_2	δ_i	$Q_i(t)$
S_0	-2	0	0.2	1.8	1.8
S_1	0	-0.3	0.3	0	1.7
S_2	0.2	0.3	0	-0.5	1.5

(c)

Figure 3. Matrix for the example task set: (a) initial values, (b) after S_1 releases 0.3, (c) after S_2 asks for 0.5.

budget. Therefore, we expect that in the average case the algorithm has a much better behavior than the worst-case (less than linear).

4 Simulation experiments

4.1 Experiments setup

In order to validate the approaches presented in this paper, we have prepared two groups of experiments. In the first one, we created 5000 systems of 10 tasks each, with uniform random periods in $[50, 800]$ and average execution times randomly selected to achieve a system utilization in $[0.65, 0.85]$. We assigned each task a reservation with period equal to the task period, and budget equal to its average execution time. The schedulability of the system was verified and a Spare Pot Sporadic Server with period equal to the lowest period among all reservations was created. The budget of the Spare Pot server has been maximized keeping the entire system schedulable, according to Equation (2).

We assume clairvoyant feedback algorithm, i.e. it knows in advance the execution time of the task, and asks the supervisor for the correct budget. The supervisor checks if the request can be satisfied, and updates the corresponding reservation budget. If the request cannot be satisfied, the supervisor increments a counter of the number of saturations. We run each experiment using all five algorithms as supervisor.

In the second set of experiments, we modeled a system consisting of 4 tasks, each one executes a MPEG II decoder. The decoding times were experimentally measured by playing four different MPEG movies with mplayer. Each task is assigned an adaptive reservation, with initial budget equal to the average decoding time, and with the Stochastic Dead-beat feedback control algorithm [1]. A Spare Pot with budget equal to the maximum average execution time of the previous four was added at the highest priority. Also, three additional tasks with fixed parameters were incorporated with a lower priority. The difference with the previous experiment is that the reservations holding the varying execution time tasks should minimize the scheduling error and the increment in the requirement may be forced by the controller loop keeping the execution time higher for longer periods. The system was simulated with all five algorithms. In all cases, the amount of saturations were counted. The scheduling error was also measured for each method.

4.2 Results obtained

Figure 4 shows the results of the first set of experiments. As can be seen, the exact method has always the lower amount of saturation points, as expected. It is important to note that the intersect method has a very similar performance. Scaling and UpBound have both a good performance at low utilization factors (below 0.75) but degrade for higher ones. The Spare Pot is near the last two methods but with a higher amount of saturation points at low utilization factors and a significant improvement in the upper part.

Figure 5 shows in a bar graph the number of saturations the five different algorithms have for the second experiment. The numbers were normalized to the exact method, as it is the reference. Except for the UpBound method, all the others have the same performance. The fact that the system saturates more frequently, means that the scheduling error may be higher and this means a lower QoS, or equivalently a higher number of missed deadlines.

Figure 6 shows the behavior of the scheduling error of each method. As can be seen, all of them have a very similar performance, except the UpBound one. In 7 the amount of missed deadlines normalized to the exact method is represented with a bar graph. Again, it is clear that the UpBound method has the poorest performance.

Finally, Figure 8 shows the total utilization factor of the system for each method. The minimum, average, maximum and the actual one for each frame of the movies is represented for each method. As can be seen, all of them behave similarly except the UpBound method.

4.3 Results explained

The results obtained show clearly the performance of the different methods.

In these experiments, the exact method acts as a reference optimal algorithm. Therefore, we are interested in evaluating the performance of the other four methods with respect to the exact. The intersect method shows very

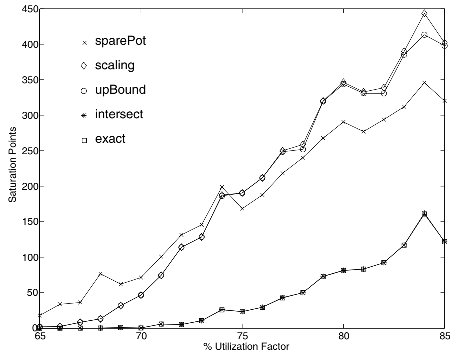

Figure 4. Saturation in random variations.

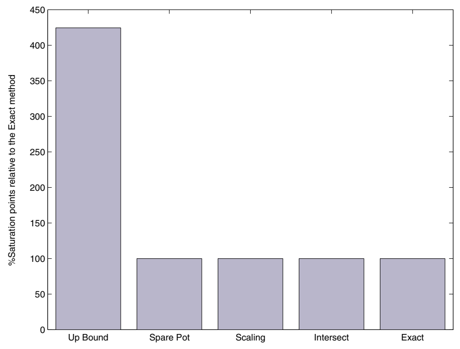

Figure 5. Saturation with feedback control

good results in both experiments. Especially in the first experiment, it is clear that its performance cannot clearly be distinguished from the exact method. This means that the algorithms selects almost all *important* scheduling point from the complete (exponential) set of points. However, it has quite an high complexity of $O(n^2)$, which makes it not the best choice for an on-line check.

The upBound algorithm shows the worst results in both experiments. Although it has constant complexity $O(1)$, in many cases its performance may not be considered enough good for the job. By looking at the scheduling error of Figure 6, it is possible to graphically evaluate the bad behavior of the algorithm.

Finally, the Spare Pot method and the scaling show good results in both experiments. In the second experiment, the performance of these algorithms are the same as the exact and intersect methods, however they have linear (or sub-linear) complexity. In the first experiment, the Spare Pot is the best of the two.

Which method to choose? Probably, the answer depends on the specific application/system. If overhead is not a concern (for example because of a low number of reservations in the system), then the intersect method is the preferred choice. When a trade-off must be made between cost and

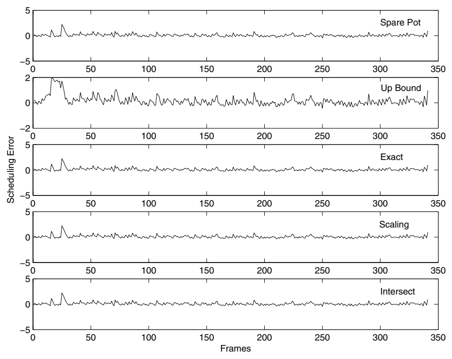

Figure 6. Scheduling error for the five methods compared

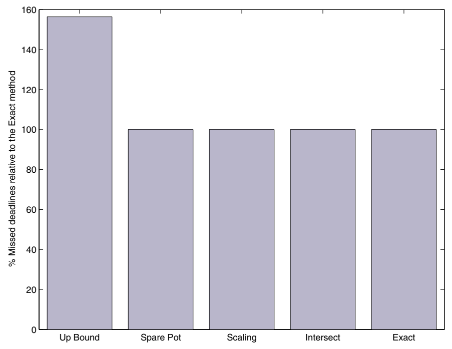

Figure 7. Missed deadlines

performance, the Spare Pot or the Scaling methods are good candidates. Finally, if the overhead is critical (for instance, when a task has a very short period), then we must sacrifice performance and select the UpBound method.

5 Conclusion and future work

In this paper the problem of managing the spare bandwidth in the system to handle the budgets of adaptive reservations through feedback control under fixed priority scheduling has been addressed. Five different solutions to this problem have been presented and compared. The solutions have different performances and complexities in their implementations. Four of them are simplification of the exact schedulability test based on the Scheduling Points method. The fifth algorithm is a novel algorithm presented in this paper, the Spare Pot, which has been proved correct. The experiments conducted over random varying sets of tasks and on a particular application like MPEG decoding, show that this solution is quite robust specially at high utilization factors.

The proper choice of the method depends on the kind of

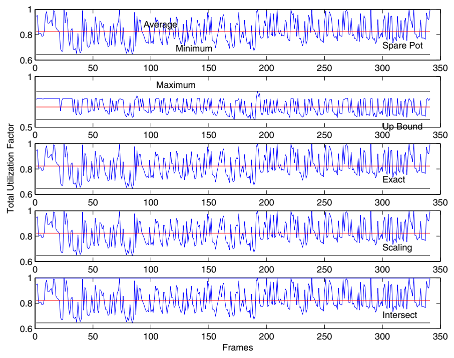

Figure 8. Total utilization factor achieved

system to be controlled and the computational power of the hardware platform.

As future work, these algorithms will be implemented in the FRESCOR framework to evaluate them under real loads. Also, a careful evaluation of the real computational requirements should be done because the theoretical bounds are rarely meet in actual implementations.

References

[1] L. Abeni, T. Cucinotta, G. Lipari, L. Marzario, and L. Palopoli. Qos management through adaptive reservations. *Real-Time Systems*, 29(2-3):131–155, 2005.

[2] L. Abeni, L. Palopoli, and G. Buttazzo. On adaptive control techniques in real-time resource allocation. In *Proceedings of the* 12th *Euromicro Conference on Real-Time Systems*, pages 129–136, Stockholm, Sweden, jun 2000.

[3] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole. Analysis of a Reservation Feedback Scheduler. In *Proc. 23rd IEEE Real Time Systems Symposium*, 2002.

[4] N. C. Audsley, A. Burns, M. Richardson, K. W. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, Sept. 1993.

[5] G. Bernat, I. Broster, and A. Burns. Rewriting history to exploit gain time. In IEEE, editor, *Proceedings of the 25th IEEE Real-Time Systems Symposium*, pages 328–335, December 2004.

[6] E. Bini and G. C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, Nov. 2004.

[7] E. Bini, M. Di Natale, and G. C. Buttazzo. Sensitivity analysis for fixed-priority real-time systems. *Real-Time Systems*, Apr. 2007. DOI: 10.1007/s11241-006-9010-1.

[8] G. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency*. Springer, 2005.

[9] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic Scheduling for Flexible Workload Management. *IEEE Transactions on Computers*, 51(3):289–302, 2002.

[10] M. Caccamo, G. Buttazzo, and L. Sha. Capacity sharing for overrun control. In *Proceedings of the* 21st *IEEE Real-Time Systems Symposium*, pages 295–304, Orlando (FL), U.S.A., Dec. 2000.

[11] M. Caccamo, G. C. Buttazzo, and L. Sha. Handling execution overruns in hard real-time control systems. *IEEE Trans. Computers*, 51(7):835–849, 2002.

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

[13] D. Chen, A. K. Mok, and T.-W. Kuo. Utilization bound revisited. *IEEE Transaction on Computers*, 52(3):351–361, Mar. 2003.

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

[15] C.-G. Lee, L. Sha, and A. Peddi. Enhanced utilization bounds for QoS management. *IEEE Transactions on Computers*, 53(2):187–200, Feb. 2004.

[16] J. P. Lehoczky, L. Sha, and Y. Ding. The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the* 10th *IEEE Real-Time Systems Symposium*, pages 166–171, Santa Monica (CA), U.S.A., Dec. 1989.

[17] G. Lipari and S. Baruah. Greedy reclaimation of unused bandwidth in constant bandwidth servers. In *Proc. 12th Euromicro Conference on Real Time Systems*, 2000.

[18] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, Jan. 1973.

[19] C. Lu, J. Stankovic, G. Tao, and S. Son. Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms. *Real Time Systems*, 23:85–126, 2002.

[20] T. Palopoli, L.and Cucinotta and A. Bicchi. Quality of service control in soft real-time application. In *Proc. 42nd IEEE Conference on Decision and Control*, pages 664–669, December 2003.

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

[22] R. Santos, G. Lipari, and E. Bini. Efficient on-line schedulability test for adaptive resource reservations. Technical Report 2, Scuola Superiore Sant'Anna – RETIS Lab, January 2008.

[23] M. Sjödin and H. Hansson. Improved response-time analysis calculations. In *Proceedings of the* 19th *IEEE Real-Time Systems Symposium*, pages 399–408, Madrid, Spain, Dec. 1998.

[24] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1:27–60, July 1989.