Real-Time Syst (2012) 48:601–634
DOI 10.1007/s11241-012-9156-y

On-line schedulability tests for adaptive reservations in fixed priority scheduling

Rodrigo Santos · Giuseppe Lipari · Enrico Bini · Tommaso Cucinotta

Published online: 1 June 2012

Abstract Adaptive reservation is a real-time scheduling technique in which each application is associated a fraction of the computational resource (a *reservation*) that can be dynamically adapted to the varying requirements of the application by using appropriate feedback control algorithms. An adaptive reservation is typically implemented by using an aperiodic server (e.g. sporadic server) algorithm with fixed period and variable budget. When the feedback law 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 fast, as it may be performed at each budget update, i.e. potentially at each instance of a task; yet, it must be as efficient as possible, to maximize resource usage.

In this paper, we tackle the problem of performing an efficient on-line schedulability test for adaptive resource reservations in fixed priority schedulers. In the literature, a number of algorithms have been proposed for on-line admission control in fixed priority systems. We describe four of these tests, with increasing complexity and performance. In addition, we propose a novel on-line test, called *Spare-Pot al-*

T. Cucinotta is now at Alcatel-Lucent Bell Labs, Ireland.

R. Santos (✉)
Dep. Ing. Eléctrica y Computadoras, Universidad Nacional del Sur, Bahía Blanca, Argentina
e-mail: ierms@criba.edu.ar

G. Lipari
Ecole Normale Supérieure, Cachan, France
e-mail: giuseppe.lipari@lsv.ens-cachan.fr

E. Bini
Department of Automatic Control, Lund University, Box 118, 221 00 Lund, Sweden
e-mail: bini@control.lth.se

T. Cucinotta
Scuola Superiore Sant'Anna, piazza Martiri della Libertà, 56124 Pisa, Italy
e-mail: tommaso.cucinotta@alcatel-lucent.com

gorithm, which has been specifically designed for the problem at hand, and which shows a good cost/performance ratio compared to the other tests.

Keywords Resource reservation · Adaptive scheduling · Fixed priority

1 Introduction

Embedded computing has emerged as an important engineering discipline with principles rooted in electronics, real-time operating systems, software engineering. Many embedded systems, including DVD and media players, digital TVs, teleconferencing systems, video servers, VoIP, etc. can be categorized as soft real-time.

Real-time systems can be roughly categorized in hard (no deadlines can be missed), soft (the number of missed deadlines influences the delivered Quality of Service) and non real-time (just best effort). Soft real-time systems require a timely behavior, but some deadlines may be missed without compromising the correctness of the results (Buttazzo et al. 2005). For example, in a pipeline of tasks that processes a video stream, it is possible to use intermediate buffers to store frames that must be processed by a late task. In this way, data is preserved in case of missed deadlines, although the end-to-end delay increases. However, the number and severity of deadline violations may have a negative impact on the Quality of Service (QoS) experienced by the user and on the amount of memory buffers that must be available. A large amount of buffers also implies more memory and this represents an important cost in embedded systems. There is a trade-off then between the deadline miss toleration, QoS, cost and user appreciation. Therefore, designers of soft real-time embedded systems must deal with two contrasting requirements: reduce the hardware resources (processors and memory) to a minimum, while maintaining a low number of deadline misses.

Scheduling is at the core of a real-time system, as it is the activity in charge of guaranteeing the timely behavior of the applications. There are many scheduling disciplines but most of them work with completely defined systems in which the number of tasks is fixed and each task is characterized by known parameters: worst-case execution time, period, deadline and release time. When some of these parameters are not known, or using the worst case would produce a very pessimistic approach, the use of the Resource Reservation Framework (RRF) (Rajkumar et al. 1998) is considered appropriate. In this framework, each task is associated with a *reservation* characterized by a budget Q and a period P. The soft real-time tasks associated to a reservation execute under a *virtual processor* with a speed proportional to the ratio between the budget and the period of the reservation Q/P. This approach can be seen as a transformation of a periodic task into a sporadic one in which the worst case execution time is no larger than Q and its minimum inter-arrival time is equal to the period of the reservation independently of the actual parameters of the task. Before activating a reservation, an admission control mechanism checks if the system continues to be schedulable. When the system is schedulable, the RRF provides the important property of *temporal isolation* by which the task scheduled by means of a reservation depends only on the reservation parameters to meet its timing constraints and not on

the presence of other tasks in the system. Examples of such algorithms are the Constant Bandwidth Server running on top of Earliest Deadline First (CBS+EDF); and the Sporadic Server running on top of Fixed Priority (SS+FP). A complete interface to RRF has been developed by the partners of the FRESCOR EU project.[1]

The budget assignment is critical for the good performance of the system; thus, a careful profiling of the computational requirements of the task is necessary. In the case of soft real-time tasks with large variations of computational requirements, a static allocation may not be enough to guarantee a good performance. In some situations the budget will be excessive with the consequence of an under-utilization of the system or, what is worst, not enough bandwidth for the rest of the tasks. On the other end, the budget could be too small and the task will not fulfill its requirements. In both cases the performance of the system is not satisfactory. Large variations in the computational requirements of a task are caused by different amount of information to be processed in each instance of the task. For example, the coding/decoding time of an I frame in a mpeg video is much longer than the required by a P frame which is longer than the required by the B frame.

To deal with this problem, the concept of *adaptive resource reservation* (ARR) (Abeni et al. 2000; Lu et al. 2002) has been introduced. In an adaptive reservation the budget follows the demand of the task by means of feedback and prediction mechanisms. Basically, each reservation has 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 control variable is the *scheduling error* defined in Abeni et al. (2005) as the difference between the actual finishing time of an instance of the task and its deadline. A positive error, means the task has violated the deadline, or, equivalently, that the assigned budget was insufficient. If the error is negative, the task completed before its deadline and the budget was sufficient. In the first case, the budget should be incremented while in the second case, it may be decremented to free resources for the other tasks. The control law should keep the scheduling error as close as possible to zero. It is important to highlight that, in this model, the change of budget is transparent to the task: in other words, the task parameters (computation time, period, etc.) do not depend on the assigned budget.

At the system level, the scheduler must guarantee that all reservations are schedulable, i.e. each instance of the reservation always completes within the reservation's period. This means that upon any modification of the budgets, dictated by the feedback controllers, the system schedulability must be checked. In particular, it might be the case that the increment determined by the controller can not be accommodated because the spare bandwidth in the system is not enough. For example, the feedback control module dictates that the budget of a reservation must be increased by ΔQ. Before incrementing the budget, a **on-line schedulability test** should be run to ensure that by incrementing the budget all reservations remain schedulable. If the test is not passed, the budget should be incremented to the maximum compatible with the schedulability of the system; in this case, it is said that the reservation is *saturated*. Since the test should be executed at each invocation of the control algorithm, it has to be as simple as possible, otherwise the overhead associated with it would be so

[1] http://www.frescor.org.

high that it will take a considerable portion of the processor bandwidth, nullifying the benefits of adaptive reservations.

Many adaptive reservation schemes have been proposed in the past, mostly in the context of EDF scheduling (Abeni et al. 2002; Palopoli et al. 2003a, 2008; Stankovic et al. 1998; Lu et al. 2002). However, little attention have been dedicated to adaptive reservation in fixed priority (FP) scheduling. Most RTOS provide POSIX compatibility, and the POSIX real-time profile provides FP and the Sporadic Server as standard scheduling disciplines. It is therefore important to study the applicability of adaptive reservation scheme on POSIX compliant RTOS (e.g. the Linux OS). In particular, it is important to study an efficient on-line schedulability test to prevent the adaptive reservation algorithm from overloading the system.

1.1 Contributions of this paper

This paper studies the problem of performing on-line schedulability test for adaptive reservations in the context of fixed priority scheduling. First, the adaptive reservation framework is proposed under fixed priorities. In particular, we will propose the use of the Sporadic Server on a Fixed Priority scheduler; none of the previous papers proposed this solution. In our framework, the schedulability test is encapsulated in a module called *Supervisor* that, in addition to performing an efficient on-line schedulability test, can be configured to implement specific allocation and mode change policies.

For the purpose of on-line admission control and adaptive reservations, we propose a family of schedulability tests based on the selection of a subset of the scheduling points (Lehoczky et al. 1989; Bini and Buttazzo 2004) (the notion of scheduling points is recalled later in Sect. 4.1). Also, we propose a novel on-line test, called *Spare-Pot*, specifically designed for dealing with adaptive reservations. Finally, we will compare the performance of all the aforementioned algorithms with simulation experiments. There is a trade-off between complexity, speed and accuracy for the schedulability evaluation after a variation in the system load. The experiments show how each of the methods proposed perform and the computational cost of them so the designer can resolve the trade-off. We will also compare with similar adaptive reservations implemented on top of CBS+EDF. The results show that the Spare-Pot algorithm is a good compromise between performance and complexity.

2 Related works

Many algorithms based on Resource Reservations have been introduced in the past. Each reservation is assigned a period and a budget, and the relation between these parameters is the bandwidth reserved. Examples of resource reservation scheduling can be found in Abeni and Buttazzo (1998), Lipari and Baruah (2000), Marzario et al. (2004) and Rajkumar et al. (1998). Resource reservations algorithms provide the property of temporal isolation, which consists in protecting every application from the possible overloads of the others.

In previous approaches, the bandwidth assignment for each reservation was based on heuristic approaches, like the average execution time of the task. However, this approach may not be optimal since very often the actual execution times differ greatly from the average ones. *Adaptive reservations* are then a natural solution to the problem. In this sense, several algorithms have been proposed in the past. For example, based on a precise dynamical model of a resource reservation scheduler, a feedback law consisting in the switching of two traditional Proportional plus Integral (PI) controllers is presented in Palopoli et al. (2003a). 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 Palopoli et al. (2003b). Additional control approaches based on both deterministic and stochastic control goals are presented in Palopoli et al. (2003b) and Cucinotta et al. (2004b). The controllers are complemented by the use of a moving average filter to predict the execution time. Formal proof of stability in the stochastic sense for generic families of controllers is provided in Cucinotta et al. (2004a). All these papers, work under Earliest Deadline First.

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, can be found in Buttazzo et al. (2002) and Caccamo et al. (2000a).

In Block et al. (2008), the authors introduced an adaptive feedback-controlled global EDF scheduling algorithm for multiprocessors soft real-time systems where the utilization of each task is time-variant and not known a priori. As the adaptive feedback-controlled scheduling is based on dynamic priorities, the approach is completely different to the one proposed here, besides the framework is oriented to multiprocessor systems.

The use of adaptive reservations requires fast on-line schedulability tests to check the feasibility of the system. In fact, it is not possible to increment the budget of a reservation if by doing it, another task becomes unschedulable. In this paper we present five different on-line schedulability tests for fixed priority real-time scheduling. Our techniques are adaptations of well-know results to the problem at hand. The literature on sufficient schedulability tests is very rich: we report here the most relevant contributions. Burchard et al. (1995) proposed a linear time utilization upper bound, which was demonstrated to be tighter than the one of Liu and Layland (1973). Han and Tyan (1997) proposed then a $o(n^2 \log n)$ test superior to the two previously mentioned. Lauzac et al. (2003) proposed a utilization upper bound to be evaluated after a $o(n \log n)$ transformation of the task periods is applied. Bini et al. (2003) proposed the hyperbolic bound: a linear time test that is strictly superior to the Liu and Layland bound. In more recent years, Bini et al. (2009) proposed a linear time test by approximating the interference of higher priority tasks with a linear function. Masrur and Chakraborty (2011) proposed to approximate the interference by a piecewise linear function.

The authors of Almeida et al. (2007) propose to use a dynamic reconfiguration method. In their model, a system is a collection of distributed subsystems interacting through a time-triggered network. The system can be in one of several modes, and in each mode a task can have different parameters. To allow fast switching between

modes, the authors propose a mix of off-line analysis (to compute the Local Utilization Bound) and dynamic analysis which simply uses the LUB calculated off-line to decide which of the many configurations is feasible on-line. Compared to our approach, the model in Almeida et al. (2007) only considers discrete modes, while the budgets of our reservations can vary on a continuous domain; also, the schedulability test proposed in Almeida et al. (2007) consists of a linear bound, and it is comparable to method upBound described in Sect. 4.1.3.

3 System model

In this section we introduce the system model used in the paper together with the related definitions. We then describe the general architecture of the system and its components. The section is concluded by a simple example that motivates our work.

An *open system* is one in which tasks may join or leave dynamically. When a task wants to join the system, it negotiates a *contract*, specifying an execution budget Q and a period P. The contract negotiation consists of an *admission control* test: if a periodic server with an execution budget of Q every P instant can be scheduled in the system without compromising the guarantee on the existing servers, the negotiation is successful, the task can be admitted in the system, and a reservation $S_i = (Q_i, P_i)$ is created to serve the task. If the test fails, the task is rejected.

The ratio $\frac{Q_i}{P_i}$ determines the utilization factor or bandwidth, $U_i = \frac{Q_i}{P_i}$. The reservations are implemented by means of appropriate server algorithms that are scheduled in fixed or dynamic priority environments, for example the sporadic server algorithm (Sprunt et al. 1989) with fixed priority (SS-FP) or the constant bandwidth server with dynamic priority (Abeni and Buttazzo 1998) (CBS-EDF).

In this paper, we analyze fixed priority scheduling, and in particular the Sporadic Server (Sprunt et al. 1989). This algorithm has been included in the real-time profile of the POSIX standard, and hence is readily available in many real-time operating systems that are POSIX compliant. Therefore, our technique can be readily applied into many existing commercial systems.

The reservation deadlines are assumed to be equal to the period (*implicit deadline model*). This simplification is made for ease of presentation, however the proposed methods are also valid when a reservation is assigned a relative deadline less than the period (*constrained deadline model*).

The admission control consists of a schedulability test: the reservations are treated as sporadic tasks with worst case execution times equal to Q_i and period equal to P_i. The negotiation is a procedure that is seldom executed in the system because the rate at which tasks ask to join or leave the system is very slow compared to the frequency of the tasks in the system. Also, the negotiation process is usually non-urgent, so it is often implemented by an appropriate *service task*[2] with its own reservation that can perform a complex schedulability test, like Response Time Analysis (Joseph and Pandya 1986; Audsley et al. 1993) or Hyperplane analysis (Bini et al. 2007). If

[2]See the FRESCOR API implementation at http://www.frescor.org.

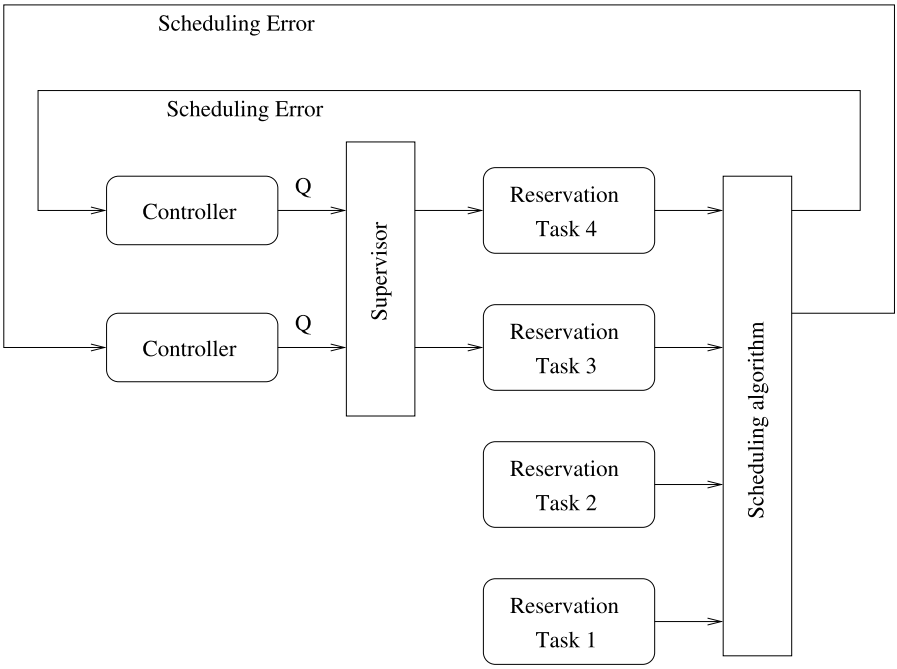


Fig. 1 Architecture of the system

admitted, the new reservation is created and guaranteed to execute Q_i unites of its budget within the reservation period P_i.

In this paper we assume that each reservation only serves one single task; extensions to hierarchical scheduling (where many tasks may be served by one reservation) is out of the scope of this paper.

3.1 Adaptive reservations

Reservations may be *fixed* or *adaptive*. A *fixed* one keeps the budget constant along the entire life of the reservation. Conversely, an *adaptive* reservation changes dynamically the budget according to the demand of the task and the free bandwidth in the system. The budget may be changed quite frequently, even at every reservation period, provided there is enough available bandwidth in the system to accommodate for such a change.

The general architecture of the system is shown in Fig. 1. A *feedback controller* is associated to each reservation. It monitors the performance of the associated task inside the reservation, and dynamically computes the minimum budget that suffices for the task to keep its ideal behavior, or restore it in case of deviations. Basically, the controller tries to keep the *scheduling error* (Abeni et al. 2005) as close to zero as possible, by measuring how late or early a task completes with respect to its soft deadline.

The use of such feedback mechanisms may seem overly complex for soft real-time systems. However, soft real-time is not synonymous of "unimportant". On the

Table 1 System's description. Minimum, average and maximum bandwidth requirements

	$U_{\min}$	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

contrary, the objective of most soft real-time is to optimise resource usage, at the same time minimising the number of deadline misses. This problem is important and complex, as discussed in Lu et al. (2002), and as testified by the rich literature on the subject (see Sect. 2).

3.2 Supervisor

Each feedback controller makes its requests independently of the others. The allocated bandwidth should never exceed the schedulability bound, otherwise some reservation could miss its timing constraint (i.e. may not be able to execute its entire budget within the period). For this reason, the requests of the controllers are evaluated by a *supervisor* that performs a quick schedulability analysis to validate the request for budget change. In case the request can not be accommodated, the reservation is saturated (i.e. the budget is set to the maximum possible value compatible with the schedulability test).

Of course, to keep the overhead of this approach as low as possible, both the schedulability analysis performed by the supervisor and the feedback control law implemented by the controller must be simple and efficient. In fact, in the worst-case, the feedback controller and the supervisor routine are 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.

The supervisor can also implement specific allocation policies: for example, each reservation can be assigned minimum and maximum budgets and the supervisor can enforce that the actual budget always stays within the limits. Also, the supervisor can implement mode change policies: for example, when a new reservation wants to join the system, the supervisor can reduce the current budgets of existing reservations to make space for the new one. This procedure will be better explained in Sect. 4.3.

3.3 Motivation for feedback scheduling

A system with four tasks is presented in Fig. 1. To simplify the presentation let us suppose that all reservation periods are equal or multiple of each other. Under this assumption, the schedulability test boils down to the well known Liu and Layland (1973) with utilization bound equal to one, $\sum U_i \leq 1$. Two tasks require a constant bandwidth of 0.25, so they negotiate two reservations with the same bandwidth, S_1 and S_2. The other two tasks demand a variable bandwidth. In both cases the minimum, average and maximum requirements are 0.1, 0.2 and 0.4, respectively. All parameters are summarized in Table 1.

With the admission of the first two tasks, only 0.5 of the bandwidth remains available. Henceforth, it is not possible to allocate the maximum requirements to the other two tasks, as this would imply a total bandwidth of 1.3, greater than the Liu and Layland bound. Allocating the minimum bandwidth is not a good option, because the tasks will usually demand more. Another option consists in allocating a fixed bandwidth equal to the average values, but this again, may not be an optimal solution, because quite often the reservation will not be able to satisfy the soft real-time task requirements, and the task deadline will probably be missed. Even increasing the fixed allocation for both tasks to 0.25 (thus reaching full utilization) may not be enough to accommodate the QoS requirements of the tasks.

The solution is to dynamically adapt the budget of the reservations corresponding to tasks 3 and 4, thus taking advantage of the statistical multiplexing property. This means that, being tasks 3 and 4 independent, it is very unlikely that they require their maximum bandwidths at the same time, hence adapting dynamically the allocation to their instantaneous expected requests will allow for accommodating both of them.

3.4 Comparison with spare reclamation algorithms

One alternative solution to the previous example is to make use of techniques for reclaiming the spare capacity. For example, task 3 requires less than its average, the extra bandwidth can be reclaimed by task 4.

Reclaiming techniques can be divided into *static* and *dynamic*. In *static reclaiming*, the excess bandwidth in the system is statically distributed across all application according to some strategy. Such a mechanism was proposed in the FRESCOR project (Zabos et al. 2009). Static allocation cannot deal with extra bandwidth due to tasks executing less than expected. Moreover, the allocation is still static, and may not be sufficient to accommodate large variations in execution time (as experienced, for example, in many multimedia applications).

Dynamic reclaiming tracks the execution times of all tasks, and can take advantage of spare bandwidth stemming from tasks that execute less than expected, or by non-allocated bandwidth in the system. Examples of such algorithms are the GRUB algorithm (Lipari and Baruah 2000), CASH (Caccamo et al. 2000b) and BASH (Caccamo et al. 2005). These algorithms redistribute the reclaimed bandwidth in a greedy manner, or divide it among all active tasks according to some static weight. Since these algorithms are greedy, a non-needing task may be assigned a large amount of extra bandwidth.

In the framework proposed in this paper, the spare bandwidth is dynamically assigned to the tasks that most need it *when* when they need it. The feedback control can in some cases *anticipate* the needs of a task thanks to application specific prediction strategies. Also, since feedback scheduling is rooted in control theory, under certain assumptions it is possible to analytically derive properties of the system like stability (Cucinotta and Palopoli 2007), convergence, response time, etc.

A more specific comparison between dynamic reclamation and feedback control techniques can be found in Palopoli et al. (2008) and Cucinotta et al. (2011), where the two techniques have been integrated, in order to take advantage of both.

4 The supervisor

The *supervisor* is perhaps the most important component of the framework. It is in charge of checking that the requests made by the feedback controllers can be accommodated by running a quick 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 latter case, the test additionally indicates the maximum allowed budget increment (*saturation*).

The schedulability test is at the heart of the supervisor and constitutes its most important part. In this paper, many different tests for fixed priority scheduling, at different levels of complexity, are evaluated. First, simple utilization bounds like the ones proposed in Liu and Layland (1973), Chen et al. (2003) and Lee et al. (2004). Then, more complex tests like the ones based on the concept of Scheduling Points (Lehoczky et al. 1989; Bini and Buttazzo 2004). Third, the Response Time Analysis (RTA) (Joseph and Pandya 1986; Audsley et al. 1993) which provides the worst-case response time of each task in the system is considered. Based on it, a new approximate schedulability test specifically devised for the problem of feedback scheduling is presented. All these have different performances and different complexities. Later, in Sect. 5, they are compared from a performance point of view and the trade-off between complexity and performance is evaluated.

4.1 Scheduling Points algorithms

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

Theorem 1 (From Lehoczky et al. (1989)) *A set of reservations* $\{(Q_i, P_i)\}$ *is schedulable under FP* ***if and only if***

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

where schedP_i *is the set of* scheduling points (Lehoczky et al. 1989; Manabe and Aoyagi 1998; Bini and Buttazzo 2004).

Since we are interested in modifications of the reservation bandwidth, we prefer to write Eq. (1) in a way such that the relationship between the bandwidth is explicit. By introducing the logical AND ($\wedge$) and the logical OR ($\vee$) operators and a more compact vectorial notation $\mathbf{U} = (U_1, \ldots, U_n)$, Eq. (1) can be rewritten as

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

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

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

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} (Manabe and Aoyagi 1998; Bini and Buttazzo 2004). In Lehoczky's initial formulation (Lehoczky et al. 1989) the set schedP_i was required to contain all the multiples of the periods P_j in $[0, D_i]$, with priority higher than τ_i, plus the deadline D_i. Formally, the initial Lehoczky definition (Lehoczky et al. 1989) of schedP_i was

$$\mathsf{schedP}_i = \left\{ r\, P_j : j = 1, \ldots, i-1, r = 1, \ldots, \left\lfloor \frac{D_i}{P_j} \right\rfloor \right\} \cup \{D_i\} \tag{3}$$

Later, Manabe and Aoyagi (1998) reduced the number of scheduling points. Bini and Buttazzo (2004) provided a recurrent definition of the reduced set of scheduling points. Thanks to these results schedP_i can be set equal to $\mathcal{P}_{i-1}(D_i)$, where $\mathcal{P}_i(t)$ is defined as follows

$$\begin{cases} \mathcal{P}_0(t) = \{t\} \\ \mathcal{P}_i(t) = \mathcal{P}_{i-1}(\lfloor \frac{t}{P_i} \rfloor) \cup \mathcal{P}_{i-1}(t) \end{cases} \tag{4}$$

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 (Bini et al. 2007) it follows that the amount of admissible variation ΔU_k to the bandwidth of the reservation S_k is

$$\Delta U_k^{\mathsf{exact}} = \min_{i=k,\ldots,n} \max_{t \in \mathsf{schedP}_i} \frac{t - \boldsymbol{\alpha}(i,t) \cdot \mathbf{U}}{\alpha_k(i,t)} \tag{5}$$

where $\alpha_k(i,t)$ denotes the kth component of the vector of coefficients $\boldsymbol{\alpha}(i,t)$. We refer to the evaluation of $\Delta U_k^{\mathsf{exact}}$ by means of Eq. (5) 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 (Bini and Buttazzo 2004), 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

$$U_1 + \frac{8}{5} U_2 \leq 1 \quad \text{from } t = 5 \tag{6}$$

$$\frac{5}{4} U_1 + U_2 \leq 1 \quad \text{from } t = 8 \tag{7}$$

and to represent them graphically (see Fig. 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. Equation (5) allows to compute the maximum admissible variation to each utilization starting from the initial point. For the values of this example we have:

$$\Delta U_1^{\mathsf{exact}} = \frac{2}{5} = 0.4 \tag{8}$$

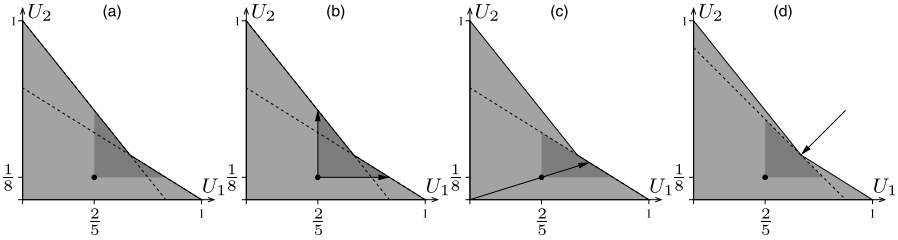


Fig. 2 Scheduling Point based methods. (**a**) Exact. (**b**) Intersect. (**c**) Scaling. (**d**) UpperBound

$$\Delta U_2^{\text{exact}} = \frac{3}{8} = 0.375 \tag{9}$$

Although for the two tasks case the computation of the maximum variation is very simple, as the number of tasks grows the evaluation of Eq. (5) 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 following result allows us to simplify the computation of the admissible variation:

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

$$\forall i = 1, \ldots, n \;\; \exists t \in \mathsf{smallSet}_i \quad Q_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{P_j} \right\rceil Q_j \leq t \tag{10}$$

Proof The corollary is simply proved by observing that if Eq. (10) is true, then $\forall i \quad \exists t \in \mathsf{smallSet}_i$ such that the inequality is true. Since $\forall i \;\; \mathsf{smallSet}_i \subseteq \mathsf{schedP}_i$, then such a t is also in the set schedP_i. So we have

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

which implies the schedulability of the task set (from Theorem 1). □

Corollary 1 suggests that, as we are able to 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.

4.1.1 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 $k = 1, \ldots, i$ so that the task τ_i is schedulable. For every pair (i, k), by using sensitivity analysis (Bini et al. 2007), we compute the maximum admissible variation of U_k that keeps task τ_i schedulable. Let then $t_k \in \mathsf{schedP}_i$ be a scheduling point such that the inequality (1) holds with the equal sign (we know there must exist one, otherwise U_k could be further increased). Then we add the instant t_k to the set $\mathsf{intersectSet}_i$. 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 Fig. 2(b). Starting from the initial assigned values (the black dot), the maximum variation along both the directions is computed, and the hit constraints are stored in the set of reduced number of scheduling points.

4.1.2 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 Fig. 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

$$\Delta U_1^{\mathsf{scaling}} = \frac{2}{5} = 0.4 \tag{11}$$

$$\Delta U_2^{\mathsf{scaling}} = \frac{1}{4} = 0.25 < \Delta U_2^{\mathsf{exact}} \tag{12}$$

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

4.1.3 The upBound method

The final method we propose is not based on the scheduling points, but on the idea of utilization upper bound (Liu and Layland 1973; Chen et al. 2003). We remind that the utilization upper bound for level-i tasks is the maximum $U_{\mathsf{ub}}^{(i)}$ such that the condition

$$\sum_{j=1}^{i} U_j \leq U_{\mathsf{ub}}^{(i)} \tag{13}$$

implies that the task τ_i is schedulable. The utilization upper bound can be computed by linear programming (Park et al. 1995; Lee et al. 2004). Once the utilization upper bound is computed, Eq. (13) can be used to compute the maximum admissible variation of the utilization.

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

$$\Delta U_1^{\mathsf{upBound}} = 0.325 \tag{14}$$

$$\Delta U_2^{\mathsf{upBound}} = 0.325 \tag{15}$$

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

4.2 Spare-Pot algorithm

In this section, a new algorithm named Spare-Pot is presented. It is based on the Response Time Analysis (RTA) schedulability test (Audsley et al. 1993), but greatly reduces the amount of computations required to determine if the required bandwidth of a reservation can be granted or not. In fact, as it will be shown later, the algorithm has linear complexity and can quickly manage the spare bandwidth in a SS-FP scheduling policy.

The underlying idea can be described in the following way. Suppose there are n adaptive reservations in the system, $S_1, S_2, \ldots, S_n$, in decreasing order of priority. Suppose there are also other non-adaptive reservations in the system with arbitrary priority levels. During negotiation, the schedulability of such reservations is checked using RTA: each reservation is considered as a sporadic task with worst-case computation time Q and minimum inter-arrival time P. With the RTA, the *worst-case response time* (WCRT) of each reservation is computed: it is important to notice that this may be different to the actual response time of the task.

At start up, the system reserves an additional *fake* reservation $S_0 = (Q_0, P_0)$, which we call *spare pot* (hence the name of the algorithm). This fake reservation has higher priority than all adaptive reservations. Its only purpose is to *reserve bandwidth* that can be collectively reclaimed by any of the adaptive reservations.

The algorithm has a *start-up phase* to be performed off-line; a *negotiation phase* performed when a task joins the system; and a *stationary phase* part that is executed during the normal run-time of the system. These phases are detailed in the next sections.

At run-time, an adaptive reservation may 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 one, it *borrows* a part of its future budget.

The reservations can only donate budget to lower priority ones, so a priority level i may donate to priority levels $i+1, i+2, \ldots, n$. Obviously, this means that they can only borrow from higher priority reservations. To handle this, 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.

4.2.1 Start-up phase

At system start up, the spare pot is created. This reservation will not execute any task; it is used for preventing the admission control from allocating all spare bandwidth to newly incoming reservations, and for distributing the spare bandwidth to the most

needing reservations. We denote this reservation with S_0, and its budget and period are Q_0 and P_0, respectively. P_0 can be arbitrarily small, as no actual server is associated with it. Therefore, P_0 can be typically set equal to some fraction of the smallest interval of time measurable in the system (e.g. 10 µs). Initially, no task is executing in the system, and $Q_0 = P_0$ (i.e. all bandwidth is Available). We require that Q_0 is never less than a minimum Q_0^{min} decided by the system administrator at start-up. The values for Q_0 and P_0 are decided based on the characteristics of the system. In particular, $\frac{Q_0^{min}}{P_0}$ represents the amount of bandwidth that is left free to account for variations in the bandwidth requirements of the reservations. If $\frac{Q_0^{min}}{P_0}$ is too small, in average there is not enough free bandwidth to account for temporary overloads of the other reservations. An analysis of how to choose these values depends on the characteristics of the applications: for example highly varying reservations will need a larger Q_0 to adapt; small variation can do with a small Q_0. A complete analysis of how to set Q_0^{min} and P_0 is out of the scope of this paper, and will be addressed in future work.

4.2.2 Negotiation phase

We denote with Q_i^N the *nominal budget* of reservation S_i, i.e. the budget requested when the reservation joins the system.

When a new reservation $S_n = (Q_n^N, P_n)$ requests to join the system, a negotiation algorithm is executed. We use the Scheduling Points method described in Sect. 4.1 to check the schedulability of the new system comprising all existing reservations with their nominal budget, the spare pot, and the new incoming reservation. We also use the sensitivity analysis (Bini et al. 2007), based on the Scheduling Point method, to compute the maximum fake $Q_0^N \geq Q_0^{min}$ such that the system remains schedulable.

If the new system is schedulable and $Q_0^N \geq Q_0^{min}$, then the task can be admitted into the system. Hence, the response times of all reservations are computed using the classical RTA analysis:

$$\forall i \quad \min\left\{ R_i | R_i = \sum_{j=0}^{i-1} \left\lceil \frac{R_i}{P_j} \right\rceil Q_j^N + Q_i^N \right\} \tag{16}$$

where Q_j^N is the nominal budget of the j-th reservation, and all reservations are sorted by decreasing priority. We assume that the fake reservation is the highest priority reservation, with index $j = 0$, and Q_0^N as computed in the previous step. R_i is the smallest solution of the fixed point equation.

The algorithm also prepares some variables that will be used after the reservation is admitted. Variable $\mathsf{preempt}(j, i)$ represents the number of instances of adaptive reservation S_j that can preempt reservation S_i in the worst case:

$$\mathsf{preempt}(j, i) = \left\lceil \frac{R_i}{P_j} \right\rceil \tag{17}$$

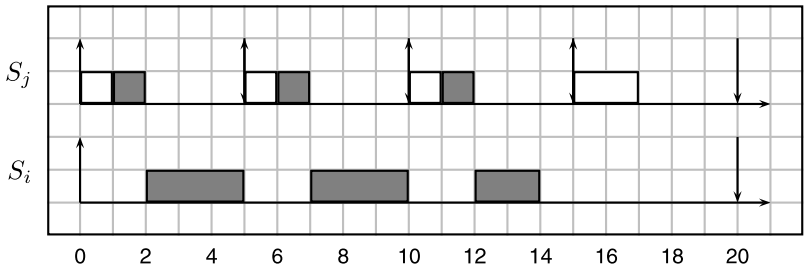


Fig. 3 Example: reservation S_j donates 3 units of execution to reservation S_i

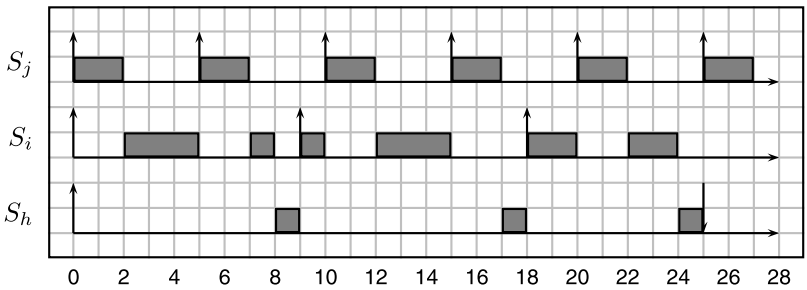


Fig. 4 Example with 3 reservations

This will be useful to compute the budget that S_j can donate to S_i without jeopardising the schedulability of the system. Of course, this variable only makes sense for $j < i$. We extend it to include the case $j = i$: $(i, i) = 1$.

In Fig. 3 we depict an example with two reservations $S_i = (Q_i^N = 8, P_i = 20)$ and $S_j = (Q_j^N = 2, P_j = 5)$. In this case, $(j, i) = 3$. Therefore, if S_j donates 1 unit of budget (the grey rectangles in the figure), reservation S_i gets $1 \cdot \mathsf{preempt}(j, i) = 3$ units of budget, and its response time will not increase. Vice versa, when S_i gives back 3 units of its budget, S_j gets 1 unit.

Unfortunately, in order to check that the system remains schedulable, we shall take into account also the other reservations in the system. Consider the example of Fig. 4 with 3 reservations $S_j = (2, 5)$, $S_i = (4, 9)$ and $S_h = (3, 25)$. In this case $\mathsf{preempt}(j, h) = 2$, so it seems possible that when S_j donates 1 unit of budget to S_i, the budget of S_i may be increased by 2. Unfortunately, doing so will make S_h miss its deadline: the resulting schedule is shown in Fig. 5, where S_h misses its deadline at $t = 25$.

To solve the problem, consider the equation for the schedulability of S_h:

$$\sum_{k=0}^{h-1} \mathsf{preempt}(k, h)\, Q_k + Q_h = R_h \leq P_h \tag{18}$$

In other words, the workload of S_h and all higher priority reservations must be equal to its response time and not greater than its period. Now, suppose we want to pass x

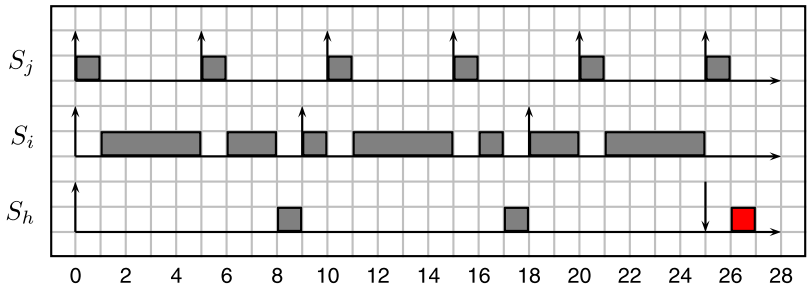


Fig. 5 If reservation S_j donates 1 unit of budget to S_i (which then receives 2 units), reservation S_h misses its deadline

units of budget from S_j to S_i. Let's compute the y units of budget that S_i can receive without modifying the response time of S_h. Then the equation becomes:

$$R_h = \sum_{k=0}^{h-1} \mathsf{preempt}(k,h)\,Q_k + Q_h - \mathsf{preempt}(j,h)x + \mathsf{preempt}(i,h)y$$

By substituting Eq. (18), we get:

$$y = \frac{\mathsf{preempt}(j,h)}{\mathsf{preempt}(i,h)}$$

In the example of Fig. 5, $x = 1$, $\frac{\mathsf{preempt}(j,h)}{\mathsf{preempt}(i,h)} = \frac{5}{3}$, therefore $y = 1.66$. It is easy to check that in this case S_h does not miss its deadline.

If we repeat this reasoning for all lower priority reservations, we can define the *reservation ratio* $\mathsf{rratio}(j,i)$ as:

$$\mathsf{rratio}(j,i) = \min\left\{\mathsf{preempt}(j,i), \min_{h>i}\left(\frac{\mathsf{preempt}(j,h)}{\mathsf{preempt}(i,h)}\right)\right\} \tag{19}$$

As before, this term only makes sense for $j \leq i$.

To keep track of how much a server S_i has received from (or given to) S_j, the algorithm maintains a matrix of $n \times n$ elements, where n is the number of reservations in the system, including the spare pot. Element (i, j) will be denoted by $\pi_{i,j}$, and has the following meaning:

- element $\pi_{i,i}$ represent the difference $\pi_{i,i} = Q_i^N - Q_i$, where Q_i^N is the nominal budget and Q_i is the current budget;
- if $i \neq j$, element $\pi_{i,j}$ represent the credit (if negative), or debit (if positive) that S_i has with S_j. In particular: if $\pi_{i,j} < 0$, then S_i has given a budget of $-\pi_{i,j}$ to S_j; if $\pi_{i,j} > 0$, then S_i has received a budget of $\pi_{i,j}$ from S_j.

Initially, all elements of the matrix are set to 0, except for $\pi_{0,0}$ that contains the Spare Pot budget Q_0^N (as Q_0 is always null). Finally, we define the *spare budget* δ_i

as:

$$\delta_i = \sum_{j=0}^{n} \pi_{i,j} \tag{20}$$

The spare budget δ_i represents the amount of budget that reservation S_i makes available to others.

4.2.3 Stationary phase

Whenever a feedback controller requires a change in the budget of reservation S_i, our algorithm is executed and the matrix is updated accordingly.

Algorithm 1 Increasing the budget

Require: $\Delta Q_i > 0$
  $j = i$
  sum $= 0$
  while $j \geq 0$ and $\Delta Q_i > 0$ **do**
    if $\delta_j > 0$ **then**
      $x_j = \min \Delta Q_i, \delta_j \mathsf{rratio}(j, i)$
      if $i \neq j$ **then**
        $\pi_{i,j} \Leftarrow \pi_{i,j} + x_j$
        $\pi_{j,i} \Leftarrow \pi_{j,i} - \frac{x_j}{\mathsf{rratio}(j,i)}$
      end if
      $\Delta Q_i \Leftarrow \Delta Q_i - x_j$
      $\pi_{i,i} \Leftarrow \pi_{i,i} - x_j$
      sum $\Leftarrow$ sum $+ x_j$
    end if
    $j \Leftarrow j - 1$
  end while
  return sum

- Suppose that the feedback controller needs to increase the budget of reservation S_i by $\Delta Q_i > 0$. The pseudo-code in Listing 1 describes what happens. The algorithm looks at δ_j, with $j = i, i-1, \ldots, 1, 0$. If $\delta_j = 0$, 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 $x_j = \min\{\Delta Q_i, \delta_j \mathsf{rratio}(j, i)\}$.

  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 $\Delta Q_i < 0$. In this case, the pseudo-code of Listing 2 is executed.

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 accordingly and $\Delta Q_i = \Delta Q_i + x_j$. 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.

Algorithm 2 Decreasing the budget

Require: $\Delta Q_i < 0$
  $\pi_{i,i} \Leftarrow \pi_{i,i} - \Delta Q_i$
  $j = 0$
  while $j < i$ and $\Delta Q_i < 0$ **do**
    $x_j = \min -\Delta Q_i, \pi_{i,j}$
    $\pi_{i,j} \Leftarrow \pi_{i,j} - x_j$
    $\pi_{j,i} \Leftarrow \pi_{j,i} + \frac{x_j}{\mathsf{rratio}(j,i)}$
    $\Delta Q_i \Leftarrow \Delta Q_i + x_j$
    $j \Leftarrow j + 1$
  end while

The following example shows how the algorithm works. The system presented in Sect. 4.1 is used. It has two tasks, τ_1 and τ_2, with $C_1 = 2$, $T_1 = 5$ and $C_2 = 1$, $T_2 = 8$ respectively. 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. Also, $\mathsf{rratio}(0,1) = \mathsf{rratio}(0,2) = \mathsf{rratio}(1,2) = 1$.

The matrix is initialised as shown in Fig. 6. The first reservation to change its budget is S_1 that releases 0.3 units of its budget. The matrix is updated accordingly and the results are shown in Fig. 7. Suppose now that reservation S_2 asks to increase its budget by 0.5. In Fig. 8(c), the new values for the matrix are shown. Notice that, in this particular case, $\pi_{i,j} = \pi_{j,i}$. However, this is not a general case as it depends on the fact that all $\mathsf{rratio}(j,i)$ are equal to 1.

Fig. 6 Initial values in the matrix for the example task set

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

Fig. 7 Matrix after reservation S_1 releases 0.3 units of budget

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

Fig. 8 Matrix after reservation S_2 asks for increasing its budget by 0.5 units

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

4.2.4 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.

Lemma 1 *Algorithms 1 and 2 guarantee that* $\forall i, \delta_i \geq 0$ *at all times.*

Proof First, observe that a reservation can only borrow from higher priority reservations, and give to lower priority ones. Therefore

$$\forall h < i \quad \pi_{i,h} \geq 0 \wedge \forall h > i \quad \pi_{i,h} \leq 0 \tag{21}$$

Algorithm 1 tries to increase the budget of reservation S_i: every time it increases some $\pi_{i,j}$, it also decreases variable $\pi_{i,i}$ of the same amount. When $j = i$ (first iteration in the loop), it only decreases $\pi_{i,i}$ if $\delta_i > 0$, and by at most δ_i. Therefore, if $\delta_i \geq 0$ when the reservation is activated, it cannot become negative by invoking Algorithm 1. Algorithm 2 tries to decrease the budget by first increasing $\pi_{i,i}$, and then giving it back to higher priority reservations. In any case, if $\delta_i \geq 0$ when the reservation is activated, it cannot become negative by invoking Algorithm 1. □

Lemma 2 *Consider a reservation* S_j. *Then*

$$\forall i > j \quad \pi_{j,j} \geq -\sum_{h=0, h\neq j}^{i} \pi_{j,h}$$

Proof By using Lemma 1 and Eq. (21), we can write:

$$\delta_j \geq 0 \qquad \pi_{j,j} \geq -\sum_{h=0}^{j-1} \pi_{j,h} - \sum_{h=j+1}^{n} \pi_{j,h} \geq -\sum_{h=0}^{j-1} \pi_{j,h} - \sum_{h=j+1}^{i} \pi_{j,h}$$

□

Theorem 2 *For any reservation* S_i, *let* Q_i *be its budget at time* t, *as modified by one or more iterations of the Spare Pot algorithm, and* Q_i^N *its nominal budget. If the reservation is fixed, then* $Q_i = Q_i^N$ *at all times. Let* R_i^N *be the nominal response time (computed during the acceptance test considering the nominal budget for all reservations) and* R_i *be the worst-case response time of reservation* S_i *computed considering each reservation* $S_j, 0 \leq j \leq i$ *to have a budget* Q_j. *Then,* $R_i \leq R_i^N$.

Proof The theorem is proved by induction. Equation (16) can be rewritten as:

$$R_i^{(k)} = Q_i + \sum_{j=0}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{P_j} \right\rceil Q_j.$$

Induction base. $\forall i,\ R_i^{(0)} = 0 \leq R_i^N$.

Induction hypothesis. $\forall i,\ R_i^{(k-1)} \leq R_i^N$.

Induction step. We will prove that $R_i^{(k)} \leq R_i^N$. By the induction hypothesis, $\lceil \frac{R_i^{(k-1)}}{P_j} \rceil \leq \lceil \frac{R_i}{P_j} \rceil = \mathsf{preempt}(j,i)$. Assuming $\mathsf{preempt}(i,i) = 1$,

$$R_i^{(k)} = \sum_{j=0}^{i} \left\lceil \frac{R_i^{(k-1)}}{P_j} \right\rceil Q_j \leq \sum_{j=0}^{i} \mathsf{preempt}(j,i)\, Q_j$$

Since $Q_i = Q_i^N - \pi_{i,i}$, we can rewrite:

$$R_i^{(k)} \leq \sum_{j=0}^{i} \mathsf{preempt}(j,i)\left(Q_j^N - \pi_{j,j}\right) = R_i^N - \sum_{j=0}^{i} \mathsf{preempt}(j,i)\pi_{j,j}$$

Now we need to prove that $\sum_{j=0}^{i} \mathsf{rratio}(j,i)\pi_{j,j} \geq 0$.

From Lemma 2, we can write:

$$\pi_{j,j} \geq -\sum_{h=0}^{j-1} \pi_{j,h} - \sum_{h=j+1}^{i} \pi_{j,h} = \sum_{h=0}^{j-1} \mathsf{rratio}(h,j)\pi_{h,j} - \sum_{h=j+1}^{i} \pi_{j,h}$$

By substituting in the previous inequality:

$$R_i^{(k)} \leq R_i^N - \sum_{j=0}^{i}\sum_{h=0}^{j-1} \mathsf{preempt}(j,i)\mathsf{rratio}(h,j)\pi_{h,j} + \sum_{j=0}^{i}\sum_{h=j+1}^{i} \mathsf{preempt}(j,i)\pi_{j,h}$$

We now swap the two summations in the second term and swap variables in the third term of the left-hand side of the inequality:

$$R_i^{(k)} \leq R_i^N - \sum_{h=0}^{i}\sum_{j=h+1}^{i} \mathsf{preempt}(j,i)\mathsf{rratio}(h,j)\pi_{h,j} + \sum_{h=0}^{i}\sum_{j=h+1}^{i} \mathsf{preempt}(h,i)\pi_{h,j}$$

By definition of rratio:

$$\mathsf{rratio}(h,j) \geq \frac{\mathsf{preempt}(h,i)}{\mathsf{preempt}(j,i)}$$

By substituting in the previous inequality:

$$R_i^{(k)} \leq R_i^N - \sum_{h=0}^{i}\sum_{j=h+1}^{i} \mathsf{preempt}(h,i)\pi_{h,j} + \sum_{h=0}^{i}\sum_{j=h+1}^{i} \mathsf{preempt}(h,i)\pi_{h,j} = R_i^N \qquad \square$$

4.2.5 *Complexity of Spare-Pot*

The complexity of this method is analysed in two different phases. During the admission control, it has the same complexity of SchedPoint. In some cases (large number of tasks and long periods), this can be very large. As anticipate, admission control is not a critical part of the system, and it can be performed by a appropriate service task.

After the tasks have been admitted, the algorithm is run at every task instance. The complexity of the method is linear in the number of reservations ($O(n)$). In the worst case, the lowest priority reservation would have to check all δ_i of the higher priority ones. Nevertheless, there are two reasons why this may be considerably lower in the average case. First, the higher priority reservations have to check for less rows of the matrix in the worst-case, so the complexity is proportional to the priority level, that is ($O(i)$). Second, it may be the case that the next higher priority reservation has enough spare budget to satisfy the demand, so the complexity in the best case is constant ($O(1)$).

4.3 Reservation dynamically joining or leaving the system

At some point, a new reservation can ask to join the system. As described in Sect. 4.2.2, the system performs admission control by using nominal budgets. If the test is successful, we have to insert the reservation in the system in which all existing reservations may have a budget that is different from their nominal one. Therefore, there may not be enough free utilisation to insert the new reservation. This problem is common to all algorithms presented till now as it depends on a structural property of our framework: spare bandwidth is re-distributed to the needing tasks. The problem is particularly relevant for the Spare Pot algorithm because all the rratio parameters may need to be changed due to the new reservation joining. Therefore, before inserting the new reservation, we have to wait for the system to return to the initial state. It is possible to make use of the supervisor to *force* all reservations to return to their nominal state, regardless of the control command. This may take at most P_{max} units of time, where P_{max} is the largest reservation period. Once all reservations return to their nominal budget, we can update the relevant variables (for example, in case of the Spare-Pot algorithm, we can add the new row into the matrix and update all rratio parameters) and then insert the new reservation in the system. Such procedure can be seen as a *mode-change*.

The fact that the controller command is ignored during the mode change causes additional errors in the control algorithms that drive the adaptive reservations. The amount of error and the consequences of such an approach depend on the application and on the feedback controller. However, the longest delay we have to wait is equal to the longest period of any reservation in the system. If all reservations have short enough periods, the problem can be limited by adding some robustness to the controllers.

5 Simulation experiments

In this section we present the results of two set of simulation experiments. The algorithms proposed in Sects. 4.1 and 4.2 are evaluated under different working condi-

tions, following two different approaches. In the first one, we tested the performance of all algorithms to follow random variations in the budget requirements for different utilization factors. The simulation was performed using a MATLAB implementation of all algorithms.

In the second approach, the general framework with the adaptive reservations controlled by feedback control algorithms and a supervisor has been implemented in RTSIM (Lipari and Bertolini). It is important to notice that RTSIM is a very complete event oriented simulator that contemplates the kernel load. Thus, the results it produces are quite close to a real implementation. In this experiment, the Spare-Pot, the Scheduling Points' Exact Method, and a fixed distribution of the bandwidth among servers were implemented and confronted to two dynamic priority approaches: CBS-EDF and CBS-EDF with hard reservation (Abeni and Buttazzo 1998; Marzario et al. 2004). The feedback control algorithm has been imported from the ARSIM library (Cucinotta and Lipari).

5.1 Saturation points

For the first approach, five thousand systems of 10 tasks each were created with random periods uniformly distributed in the interval [50, 800]. Tasks have variable execution times, and average execution times randomly selected as to give system's utilization factors varying from 0.65 to 0.85.

As explained in Sect. 4, a saturation occurs whenever a task demands a certain budget that can not be provided by the supervisor. In that case, the supervisor computes the maximum amount of budget allowed, and the reservation is said to be *saturated*.

Each task is allocated to a reservation with period equal to the task's 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 Eq. (2).

For each system, the Exact, Intersect, Scaling, UpBound and SparePot methods were run according to the following conditions.

A *clairvoyant* feedback control algorithm was assigned to each task, which knows in advance the execution time of each instance of the tasks and asks the supervisor for the correct budget to the next task instance. The supervisor checks if the request can be satisfied, and updates the corresponding reservation budget. If the reservation is saturated, a variable is incremented that counts the total number of *saturations*. This variable indicates the ability of the supervisor to satisfy the task requests, and hence to take advantage of the available spare bandwidth in the system. The more effective is the supervisor, the least the number of saturations; therefore, this variable can be considered one direct performance index of the schedulability test implemented by the supervisor.

Figure 9 shows the results of the first set of experiments. As can be seen, the exact method has always the lowest amount of saturation points, as expected. Therefore, it acts as a reference optimal algorithm. However, as we will see in Sect. 5.2, its high complexity makes it impractical for use in a on-line supervisor.

The intersect method shows very similar performance, and in most cases it is indistinguishable from the Exact method. This means that the algorithm selects almost all

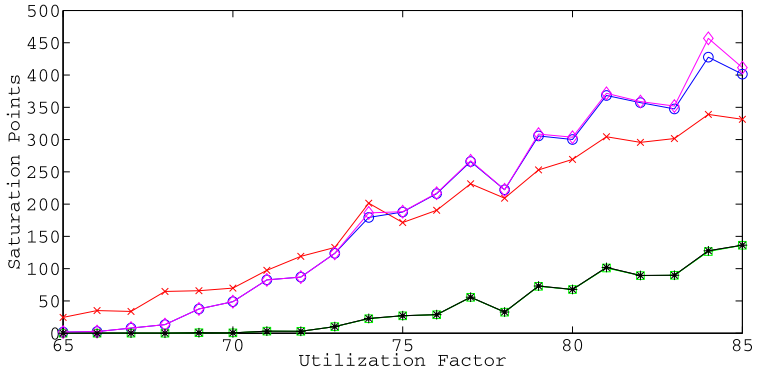


Fig. 9 Saturation in random variations. o Upper bound. × Spare Pot. ◇ Scaling. * Intersect. ⋆ Exact

important scheduling points from the complete (exponential) set of points. However, it still has quadratic complexity $O(n^2)$ in the number n of tasks in the system, which could make it unfit when n is particularly large. The upBound and Scaling algorithms show the worst results: although they show low complexity, in many cases their performance may not be considered good enough for the job, especially at high load. Algorithm Spare-Pot shows better performance under high load conditions, while it does not behave very well in small loads. One reason for this behavior may be due to the fact that in Spare-Pot a reservation can only donate to low-priority reservations: this means that if a low priority reservation executes less than its nominal budget, its extra budget cannot be reused by higher priority reservations.

5.2 Complexity

In this section, we measure experimentally the computational cost of every algorithm. In particular, we measured the number of multiplications (or divisions) required to reach the result for each one of the 5 schedulability tests presented in this paper, using the MATLAB simulator. We only measured the complexity of the stationary phase. The preparation phase is different for each algorithm and in some cases requires the solution of an optimization problem with linear programming routines like in the upBound method.

Two different approaches were followed for the evaluation of the complexity. In the first one, the amount of multiplications are measured for each different utilization factor. In the second approach, the utilization factor is fixed to 70 % and the number of tasks is varied from 10 to 50. For each system the amount of multiplications are counted and represented.

Table 2 shows the theoretical complexity, the expected amount of multiplications and the actual average amount of multiplications obtained from the simulation for the set of ten tasks. The actual amount of multiplications is lower than the expected one. The improvement in the performance comes from the fact that many scheduling points are repeated and so it is not necessary to repeat the multiplications. In particular, for the case of the Exact and Intersect methods, an important memory space should be reserved for storing the intermediate results.

Table 2 Computational costs

Method	O(.)	Expected	Simulated
Exact	$n2^n$	10240	815
Intersect	n^2	100	58
Scaling	n	10	5.5
UpBound	1		1
SparePot	n	10	4.2

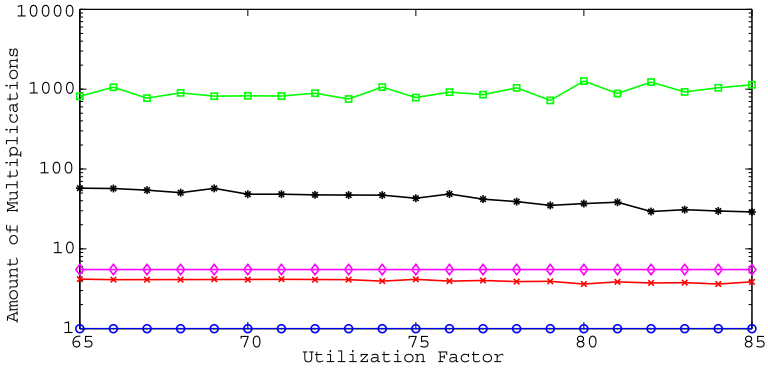


Fig. 10 Multiplications performed. o Upper bound. × Spare Pot. ◇ Scaling. * Intersect. □ Exact

Figure 10 shows the average amount of multiplications for different utilization factors. Please notice that the y axis is in logarithmic scale. The results show that the complexity is independent of the utilization factor.

Figure 11 shows the variation in the complexity as the number of tasks is incremented. Again, please notice that the y axis is in logarithmic scale. As can be seen UpBound has a constant complexity, while Scaling and the SparePot have a linear increment with the amount of tasks. In the case of Intersect and Exact methods, the complexity soon becomes much higher than the others: therefore, their use must be carefully considered depending on the number of active reservations in the system.

The selection of a method to determine the variations in the reservations capacities should be based on different aspects one of which is the complexity. Obviously, the Exact method gives the best results and the system can accommodate more variations and the amount of saturations is reduced. However, there is an important cost associated both in the preparation phase and in the stationary one. The Intersect method reduces the complexity and provides similar results to the Exact one. However, for this particular example, it still is one order of magnitude more expensive in terms of complexity than the other methods. The Scaling and Upper Bound methods are very simple, but they results are not good for an adaptive system as they easily reach the saturation. Finally, the Spare-Pot algorithm proposed in this paper, has a complexity similar to the Scaling method, and it can be a good option in terms of performance/cost ratio.

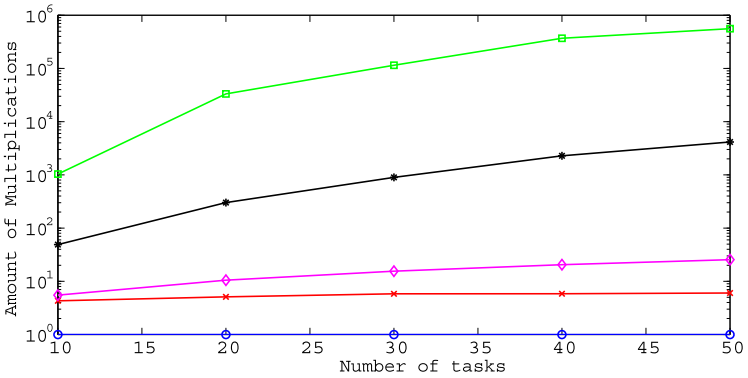


Fig. 11 Multiplications performed. o Upper bound. × Spare Pot. ◇ Scaling. * Intersect. □ Exact

5.3 Tardiness

In the second set of experiments, the entire framework was simulated with RTSIM. For the experiments, seven tasks were defined. Four of them represent a fix load while the other three represent a variable load. The last ones are named multimedia tasks, and their computation times are obtained from true video-frames decoding times of the Matrix movie. The computation times of these have a great variability and the worst case can vary between 1.5 and 4 times the average one.

In the first place, each multimedia task is scheduled through a Sporadic Server and the whole system under Rate Monotonic. The percentage of missed deadlines and the average tardiness is measured for each one of the multimedia tasks as the fixed load increments its participation in the system's utilization factor from 0.32 to 0.57. For each combination, three different policies for the budget of the sporadic servers are considered:

– In the first one, fixed reservations are considered. The amount of spare bandwidth is evenly distributed among the three multimedia servers.
– In the second and third ones, adaptive reservations are considered. In the second case, the supervisor used the Spare-Pot algorithm to handle the increment/decrement of the budgets.
– In the third case, and the one used as benchmark, the exact computation of the budgets' increments/decrements is performed following the SchedPoints method.

For the adaptive reservation cases, the bandwidth is dynamically adjusted following some kind of predictor/controller mechanism. In particular, the feedback controller is configured with a predictor estimating the maximum of the next job computation time as the maximum observed over a moving window including the last 24 jobs, and a control algorithm aiming to guarantee a probability of deadline non-violation (PDNV controller) of 1.0, based on such estimate (see Palopoli et al. 2008 for details about this control algorithm).

In the second place, the same configuration for load tasks and multimedia ones was scheduled with a dynamic priorities approach. The reservations were implemented through CBS (Abeni and Buttazzo 1998) and CBS-HR (Marzario et al. 2004). The

Table 3 System's description

Type	C_{min}	C_{max}	C_{avg}	T	p
L			–	35000	1
M	1502	14177	4250	40000	2
L			–	50000	3
M	1494	6908	4327	60000	4
L			–	70000	5
M	1493	6934	4323	80000	6
L			–	100000	7

Table 4 % M1 missed deadlines

Method	32	37	42	47	52	57
Fix	0.200	0.300	0.281	3.928	37.273	49.55
SparePot	0.492	0.492	0.492	12.866	16.799	43.734
SchedPoint	0.492	0.492	0.492	0.492	0.492	2.607
CBS-EDF	0	0	0	0	0.010	0.030
CBSHR-EDF	0.503	0.503	0.503	0.503	0.503	0.503

Table 5 % M2 missed deadlines

Method	32	37	42	47	52	57
Fix	0	0	0.45	2.113	45.241	60.264
SparePot	0.528	0.528	0.528	0.664	13.094	45.029
SchedPoint	0.528	0.573	0.573	0.573	0.604	2.49
CBS-EDF	0	0	0	0	0	0.060
CBSHR-EDF	0.528	0.528	0.528	0.528	0.573	0.573

controller was the same used for the fixed priority case. These experiments were included because EDF scheduling allows for a better utilization of the processor so it can be used as a general benchmark.

Table 3 presents the tasks priorities, periods and mean, max and minimum computation times for the multimedia tasks. For the fix load tasks only the periods and priorities are presented because their computation times will vary in the experiments to evaluate the performance of the SparePot algorithm. Multimedia tasks are marked with "M" and fix load tasks with "L".

The results for the second set of experiments are presented in Tables 4, 5, 6, 7, 8 and 9 and Figs. 12, 13, 14, 15, 16 and 17. In the first three tables, the percentage of missed deadlines for the different multimedia tasks scheduled with the five methods for the different fixed loads are presented. In the last three tables, the tardiness for the multimedia tasks are presented in the same way. The simulation time was set as to have 10000 instances for the first multimedia task, 6667 instances for the second one and 5000 instances for the third one.

Table 6 % M3 missed deadlines

Method	32	37	42	47	52	57
Fix	0	0	0	0.1	38.034	55.135
SparePot	0.543	0.543	0.543	1.420	9.601	79.376
SchedPoint	0.543	0.543	0.543	0.543	0.624	3.262
CBS-EDF	0	0	0	0	0	0
CBSHR-EDF	0.543	0.543	0.543	0.543	0.543	0.543

Table 7 % M1 tardiness

Method	32	37	42	47	52	57
Fix	0.000026	0.000115	0.000190	0.000948	0.009398	0.013706
SparePot	0.000387	0.000392	0.000398	0.000640	0.004151	0.012200
SchedPoint	0.000387	0.000392	0.000398	0.000403	0.000409	0.000963
CBS-EDF	0.000000	0.000000	0.000000	0.000000	0.000008	0.000027
CBSHR-EDF	0.000404	0.000410	0.000417	0.000423	0.000435	0.000459

Table 8 % M2 tardiness

Method	32	37	42	47	52	57
Fix	0.000000	0.000000	0.000025	0.001446	0.030512	0.043177
SparePot	0.000288	0.000310	0.000334	0.000581	0.009585	0.032271
SchedPoint	0.000288	0.000393	0.000422	0.000436	0.000460	0.002264
CBS-EDF	0.0	0.000000	0.000000	0.000000	0.000000	0.000064
CBSHR-EDF	0.000304	0.000332	0.000385	0.000426	0.000503	0.000584

Table 9 % M3 tardiness

Method	32	37	42	47	52	57
Fix	0.000000	0.000000	0.000000	0.000146	0.046148	0.067784
SparePot	0.000496	0.000511	0.000537	0.002113	0.022102	0.106333
SchedPoint	0.000992	0.001014	0.001073	0.001151	0.003592	0.025089
CBS-EDF	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
CBSHR-EDF	0.000480	0.000493	0.000517	0.000559	0.000645	0.000804

Results show that dynamic priorities methods outperform fixed priorities ones, and this is no news. However, it is important to mark here that the utilization of the Exact Scheduling Point method is not far in performance from the CBS with Hard Reservation one. Percentage of deadlines missed and tardiness are naturally in accordance. The Spare-Pot algorithm has a good behavior when the system has a middle utilization factor.

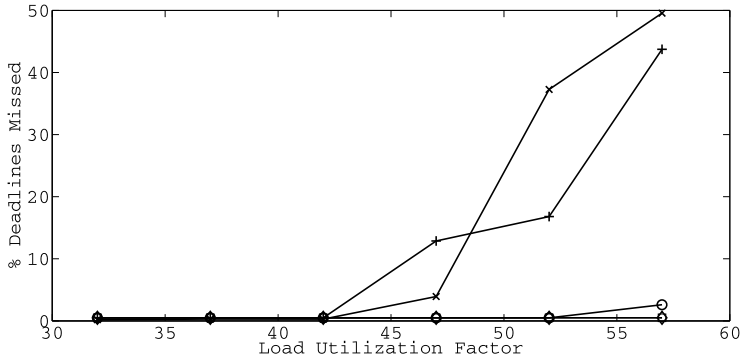


Fig. 12 M1 percentage deadlines missed. × Fix. + SparePot. o SchedPoints. ◇ CBS. ⋆ CBS-hard

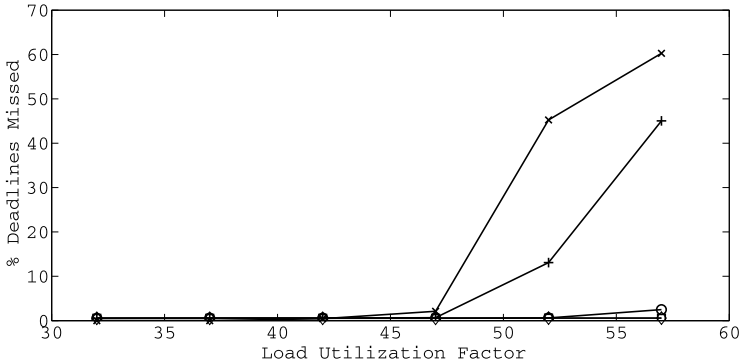


Fig. 13 M2 percentage deadlines missed. × Fix. + SparePot. o SchedPoints. ◇ CBS. ⋆ CBS-hard

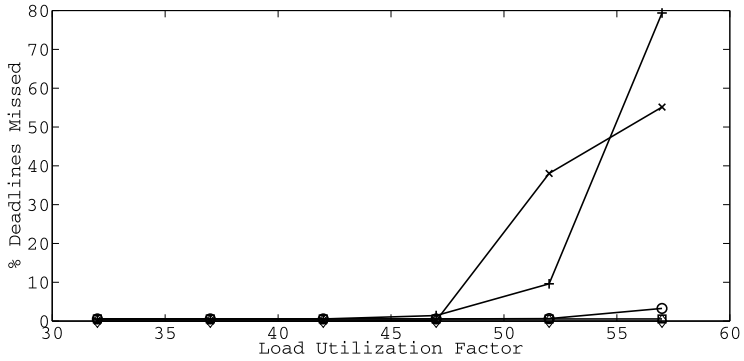


Fig. 14 M3 percentage deadlines missed. × Fix. + SparePot. o SchedPoints. ◇ CBS. ⋆ CBS-hard

6 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

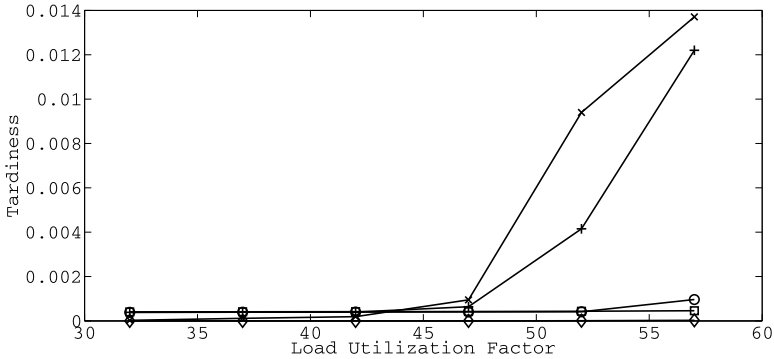


Fig. 15 M1 tardiness. × Fix. + SparePot. o SchedPoints. ◇ CBS. ⋆ CBS-hard

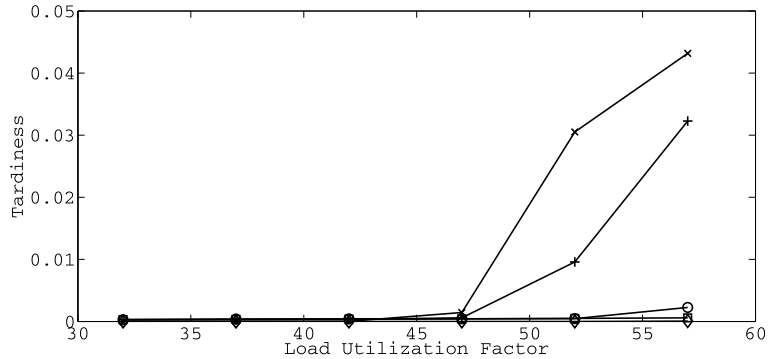


Fig. 16 M2 tardiness. × Fix. + SparePot. o SchedPoints. ◇ CBS. ⋆ CBS-hard

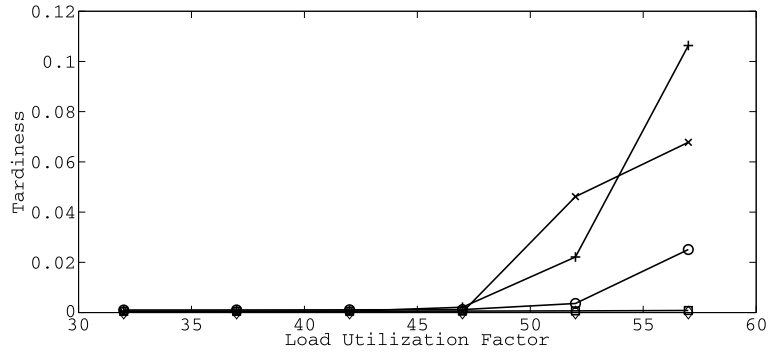


Fig. 17 M3 tardiness. × Fix. + SparePot. o SchedPoints. ◇ CBS. ⋆ CBS-hard

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 schedu-

lability 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 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

Abeni L, Buttazzo G (1998) Integrating multimedia applications in hard real-time systems. In: Proc 19th IEEE real time systems symposium

Abeni L, Buttazzo G (1998) Integrating multimedia applications in hard real-time systems. In: Proceedings of the 19th IEEE real-time systems symposium, Madrid, Spain, December 1998, pp 4–13

Abeni L, Palopoli L, Buttazzo G (2000) On adaptive control techniques in real-time resource allocation. In: Proceedings of the 12th Euromicro conference on real-time systems, Stockholm, Sweden, June 2000, pp 129–136

Abeni L, Palopoli L, Lipari G, Walpole J (2002) Analysis of a reservation feedback scheduler. In: Proc 23rd IEEE real time systems symposium

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

Almeida L, Anand M, Fischmeister S, Lee I (2007) A dynamic scheduling approach to designing flexible safety-critical systems categories and subject descriptors. In: Proc of the 7th annual ACM conference on embedded software EMSOFT

Audsley NC, Burns A, Richardson M, Tindell KW, Wellings AJ (1993) Applying new scheduling theory to static priority pre-emptive scheduling. Softw Eng J 8(5):284–292

Bini E, Buttazzo GC (2004) Schedulability analysis of periodic fixed priority systems. IEEE Trans Comput 53(11):1462–1473

Bini E, Buttazzo GC, Buttazzo GM (2003) Rate monotonic scheduling: the hyperbolic bound. IEEE Trans Comput 52(7):933–942

Bini E, Di Natale M, Buttazzo GC (2007) Sensitivity analysis for fixed-priority real-time systems. Real-Time Syst 39(1–3):5–30

Bini E, Huyen T, Nguyen C, Richard P, Baruah SK (2009) A response-time bound in fixed-priority scheduling with arbitrary deadlines. IEEE Trans Comput 58(2):279–286

Block A, Brandenburg B, Anderson JH, Quint S (2008) An adaptive framework for multiprocessor real-time system. In: Euromicro conference on real-time systems. ECRTS'08, July 2008, pp 23–33

Burchard A, Liebeherr J, Oh Y, Son SH (1995) New strategies for assigning real-time tasks to multiprocessor systems. IEEE Trans Comput 44(12):1429–1442

Buttazzo G, Lipari G, Caccamo M, Abeni L (2002) Elastic scheduling for flexible workload management. IEEE Trans Comput 51(3):289–302

Buttazzo G, Lipari G, Abeni L, Caccamo M (2005) Soft real-time systems: predictability vs. efficiency. Springer, Berlin

Caccamo M, Buttazzo G, Sha L (2000a) Elastic feedback control. In: IEEE proceedings of the 12th Euromicro conference on real-time systems, pp 121–128

Caccamo M, Buttazzo G, Sha L (2000b) Capacity sharing for overrun control. In: Proceedings of the 21st IEEE real-time systems symposium, Orlando (FL), USA, December 2000, pp 295–304

Caccamo M, Buttazzo GC, Thomas DC (2005) Efficient reclaiming in reservation-based real-time systems with variable execution times. IEEE Trans Comput 54(2):198–213

Chen D, Mok AK, Kuo T-W (2003) Utilization bound revisited. IEEE Trans Comput 52(3):351–361

Cucinotta T, Lipari G Adaptive reservation simulator. http://gna.org/projects/arsim

Cucinotta T, Palopoli L (2007) Feedback scheduling for pipelines of tasks. In: Proceedings of the 10th international conference on hybrid systems: computation and control, HSCC'07. Springer, Berlin, pp 131–144

Cucinotta T, Palopoli L, Marzario L (2004a) Stochastic feedback-based control of qos in soft real-time systems. In: 43rd IEEE conference on decision and control, 2004. CDC, vol 4, pp 3533–3538

Cucinotta T, Palopoli L, Marzario L, Lipari G, Abeni L (2004b) Adaptive reservations in a Linux environment. In: Proc of 10th IEEE real-time and embedded technology and applications symposium

Cucinotta T, Abeni L, Palopoli L, Lipari G (2011) A robust mechanism for adaptive scheduling of multimedia applications. ACM Trans Embed Comput Syst 10(4):1–24

Han C-C, Tyan H-y (1997) A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithm. In: Proceedings of the 18th IEEE real-time systems symposium, San Francisco (CA), USA, December 1997, pp 36–45

Joseph M, Pandya PK (1986) Finding response times in a real-time system. Comput J 29(5):390–395

Lauzac S, Melhem R, Mossé D (2003) An improved rate-monotonic admission control and its applications. IEEE Trans Comput 52(3):337–350

Lee C-G, Sha L, Peddi A (2004) Enhanced utilization bounds for QoS management. IEEE Trans Comput 53(2):187–200

Lehoczky JP, Sha L, Ding Y (1989) The rate-monotonic scheduling algorithm: exact characterization and average case behavior. In: Proceedings of the 10th IEEE real-time systems symposium, Santa Monica (CA), USA, December 1989, pp 166–171

Lipari G, Baruah SK (2000) Greedy reclamation of unused bandwidth in constant bandwidth servers. In: Proceedings of the 12th Euromicro conference on real-time systems, Stockholm, Sweden, June 2000

Lipari G, Bertolini C Real-time simulator. http://rtsim.sssup.it/

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard real-time environment. J Assoc Comput Mach 20(1):46–61

Lu C, Stankovic J, Tao G, Son S (2002) Feedback control real-time scheduling: framework, modeling and algorithms. Real-Time Syst 23:85–126

Manabe Y, Aoyagi S (1998) A feasibility decision algorithm for rate monotonic and deadline monotonic scheduling. Real-Time Syst 14(2):171–181

Marzario L, Lipari G, Balbastre P, Crespo A (2004) Iris: A new reclaiming algorithm for server-based real-time systems. In: IEEE real-time and embedded technology and applications symposium, pp 211–218

Masrur A, Chakraborty S (2011) Near-optimal constant-time admission control for DM tasks via non-uniform approximations. In: Proceedings of the 17th IEEE real-time and embedded technology and applications symposium (RTAS), Chicago, IL, USA, pp 57–67

Palopoli L, Abeni L, Lipari G (2003a) On the applications of hybrid control to CPU reservations. In: Proc of hybrid systems computation and control HSCC03. LNCS

Palopoli L, Cucinotta T, Bicchi A (2003b) Quality of service control in soft real-time application. In: Proc 42nd IEEE conference on decision and control, December 2003 pp 664–669

Palopoli L, Abeni L, Cucinotta T, Lipari G, Baruah SK (2008) Weighted feedback reclaiming for multimedia applications. In: IEEE/ACM/IFIP workshop on embedded systems for real-time multimedia. ESTImedia 2008, October 2008 pp 121–126

Park D-W, Natarajan S, Kanevsky A, Kim MJ (1995) A generalized utilization bound test for fixed-priority real-time scheduling. In: Proceedings of the 2nd international workshop on real-time systems and applications, Tokyo, Japan, October 1995, pp 73–77

Rajkumar R, Juvva K, Molano A, Oikawa S (1998) Resource kernels: A resource-centric approach to real-time and multimedia systems. In: Proc of the SPIE/ACM conference on multimedia computing and networking, January 1998

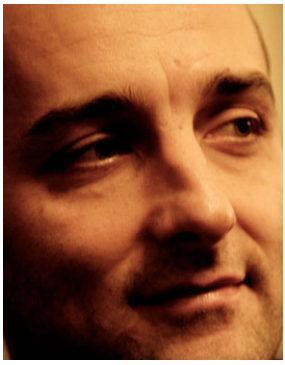
Sprunt B, Sha L, Lehoczky JP (1989) Aperiodic task scheduling for hard-real-time systems. Real-Time Syst 1:27–60

Stankovic JA, Lu C, Son SH (1998) The case for feedback control in real-time scheduling. In: Proceedings of the IEEE Euromicro conference on real-time, York, England, June 1998

Zabos A, Davis R, Burns A, Gonzalez Harbour M (2009) Spare capacity distribution using exact response-time analysis. In: 17th international conference on real-time and network systems, Paris, France, October 2009

Rodrigo Santos is Adjunt Professor at the Department of Electrical Engineering and Computers at Universidad Nacional del Sur of Computer Architectures and Networks and Adjunt Researcher at Instituto de Investigaciones en Ingeniería Eléctrica of CONICET. His research interests are centered around embedded systems and mobile communication protocols in the real-time field. In 2003, 2005, 2007 and 2010 he has been a visiting scholar at the Scuola Superiore Sant'Anna, University of Pisa, Italy. He is the President of the Latin American Center of Studies in Informatics (CLEI) (2008–2012) and Vice-chair for the Working Group 6.9 of TC 6 IFIP: Communications in Developing Countries.

Giuseppe Lipari is Associate Professor of Computer Engineering (scientific sector ING-INF/05) at Scuola Superiore Sant'Anna. He is part of the RETIS lab of the Center of Excellence for Information, Communication and Perception Engineering. He is Senior IEEE member, and associate editor of Journal of System Architecture. His research interests are in real-time systems, real-time operating systems, scheduling algorithms, embedded systems, wireless sensor networks. He has recently moved to the Ecole Normal Supérieure at Cachan France for a two year period.



Enrico Bini is Marie-Curie fellow at Lund University. He was assistant professor at Scuola Superiore Sant'Anna in 2006–2012. He achieved a Computer Engineering degree at University of Pisa in 2000, and the PhD at Scuola Superiore Sant'Anna in 2004 on Real-Time Systems. In 2010 he also achieved a second Master degree in Mathematics with a thesis on optimal sampling for control systems. He visited the University of North Carolina at Chapel Hill (10 months in 2003) and INRIA Rocquencourt (4 months in 2009).
His research interests include: scheduling algorithms, optimization, discrete mathematics, real-time operating systems, and control systems. He published more than 70 papers in journals and international conferences, won 2 best paper awards, and was invited in 36 program committees of international conferences.

Tommaso Cucinotta graduated in Computer Engineering at the University of Pisa in 2000, and received the PhD degree in Computer Engineering from the Scuola Superiore Sant'Anna of Pisa in 2004. He has been Assistant Professor of Computer Engineering at the Real-Time Systems Laboratory (ReTiS) of Scuola Superiore Sant'Anna, with research activities in the areas of real-time and embedded systems, with a particular focus on realtime support for general-purpose Operating Systems, and security, with a particular focus on smart-card based authentication. Since January 2012, he is a Researcher at Alcatel-Lucent Bell Laboratories in Dublin, Ireland.