# Dynamic Reconfiguration for Adaptive Multiversion Real-Time Systems [*]

George Lima
Depart. of Computer Science (DCC)
Federal University of Bahia (UFBA)
Salvador, BA, Brazil
gmlima@ufba.br

Eduardo Camponogara
Depart. of Automation and Systems Engineering (DAS)
Federal University of Santa Catarina (UFSC)
Florianópolis, SC, Brazil
camponog@das.ufsc.br

Ana Carolina Sokolonski
Depart. of Computer Science (DCC)
Federal University of Bahia (UFBA)
Salvador, BA, Brazil
carolsokolonski@gmail.com

## Abstract

*Modern real-time systems must be designed to be highly adaptable, reacting to aperiodic events in a predictable manner and exhibiting graceful degradation in overload scenarios whenever needed. In this context, it is useful to structure the system as a set of multiversion tasks. Task versions can be modeled to implement services with various levels of quality. In overload scenarios, for instance, a lower quality service may be scheduled for execution keeping the system correctness and providing graceful degradation. The goal of the reconfiguration mechanism is to select the versions of tasks that lead to the maximum benefit for the system at runtime. In this paper, we provide a schedulability condition based on which we derive an optimal pseudo-polynomial solution for this problem. Then, a faster approximation solution is described. Results from simulation indicate the effectiveness of the proposed approach.*

## 1 Introduction

Real-time systems, which once were characterized by having simple periodic predictable behavior, have become complex systems. Dealing with aperiodic events, fault tolerance, and the need to be highly reactive and adaptive are some of the new requirements of modern real-time systems. Consider a real-time surveillance system as an illustration. It must provide routines to detect and identify motion image patterns of moving objects. The monitored objects may activate different routines for image processing, each one providing a given quality of service. The object type, its distance, speed or the current environment luminance are examples of parameters that may trigger some appropriate routine. Fault tolerance requirements are other aspect present in modern real-time systems. Error detection may trigger recovery tasks that must execute by a given deadline. Similar adaptation requirements are also necessary for new energy-aware systems, automotive systems, autonomous robot systems etc. In any case, such modern real-time systems may work under eventual overload conditions and must be highly adaptive, ensuring temporal correctness but exhibiting graceful degradation.

Real-time scheduling is in the core of the support for adaptive applications and several approaches have been proposed to deal with overload conditions. They can be divided into classes, depending on their main goal. There are approaches that provide *temporal isolation*. They are usually based on reserving the necessary computing resources for each task or group of tasks that compose the system. If more than what was reserved is needed only part of the system may suffer the effects of overload. Server-based mechanisms [15, 20] belong to such a class. There are also approaches that aim at *protecting* the whole system against overload scenarios. Usually, the system is equipped with an

[*]This work has been funded by CNPq (grant 475851/2006-4), CAPES (PROCAD) and FAPESB (grants 7630/2006 and 3381/2005).

admission control mechanism that rejects new tasks if they may cause overload [15, 7]. Approaches that provide *dynamic reconfiguration* belong to another class and are the focus of this paper. These are mechanisms that are capable of *dynamically selecting* which parts of the system are more suitable to be executed under a given overload scenario.

One approach to dynamic reconfiguration deals with applications structured as a set of multiversion tasks. Each task has a set of versions each of which has an execution cost and provides a benefit to the system. Multiversion-like scheduling is usual for providing fault tolerance, where versions of a task are related to recovery routines [13]. This scheduling model is also in line with other application domains. For instance, there have been researches on satellite systems [19] and on energy-aware scheduling [18] that make use of multivesion tasks.

The goal of the reconfiguration mechanism proposed in this paper is to select the appropriate task versions that maximize the global benefit for the system. Clearly, this problem is not simple since this involves solving a reasonably complex optimization problem at runtime subject to the system schedulability conditions. The proposed solution to this problem is based on dynamic programming techniques. First, an optimal solution to the problem, which has a pseudo-polynomial runtime complexity, is described. We then derive a faster approximation algorithm that can be adjusted so that the designer may trade memory and speed with optimality.

The remainder of this paper is structured as follows. Section 2 provides an overview on similar work. The assumed model of computation is described in Section 3. Then, the proposed reconfiguration mechanism is explained in Section 4. This section also brings the assessment of the proposed solutions, showing results from simulation. Conclusions are drawn in Section 5.

## 2 Related Work

Perhaps the simplest form of dynamic reconfiguration is through temporal protection where the rejection or cancellation of tasks is carried out by means of admission control mechanisms [11, 3, 16, 7]. This approach may be useful for soft real-time applications due to their tolerance to missing deadlines. For some systems, however, eventual executions of some services in a degraded mode may be more appropriate. In this case, providing a more elaborate dynamic reconfiguration mechanism is required.

The main goal of a reconfiguration mechanism is to select the suitable operation modes of the system tasks in order to optimize a certain global objective function. A given quality of service is associated to each task operation mode. A configuration can be seen as an assignment of the task operation modes to system tasks. A task operation mode can be expressed by a particular operation period or a task version. In the former case, usually higher release frequency gives better quality of service. For example, in a control system, the control and sampling tasks may be released at high frequencies when the system operates in high quality mode or at low frequencies when it is running in a degraded mode. When operation modes are expressed by multiple versions of a task, each version exhibits a quality of service and requires a different execution cost. In both cases, reconfiguration is a means of adaptiveness for the system.

Some reconfiguration mechanisms that use task periods as a reconfiguration parameter have been proposed. Buttazzo and Abeni [6] have considered EDF scheduled systems. Their approach associates elastic coefficients with the system tasks. This model has been recently extended by Chantem *et al.* [8]. According to the model, the higher the coefficients the easier the period reconfiguration. The main difficulty with the elastic model is to assign coefficients to tasks since this is not an intuitive parameter from the application point of view.

In the context of Rate Monotonic (RM) scheduling, Beccari *et al.* [2] have proposed a set of heuristics, which aim at selecting, under overload conditions, task periods for non-critical tasks so that their execution benefit for the system is optimized. It is assumed that there is a benefit value associated with each soft task mode. The work by Kuo and Mok [12] is also based on RM. In order to select the task periods, the reconfiguration mechanism requires that task periods are harmonically related to each other. Also, the proposed model restricts the periods and the worst-case execution times of tasks, assuming that they are linearly corelated. These assumptions may prevent the applicability of this solution for modern real-time systems.

In this work we consider that each task has one or more versions that must be selected at runtime. Thus, instead of considering multiple task periods, the reconfiguration parameter is the worst-case execution time of each task version. We consider in the model periodic and aperiodic tasks. Therefore, fault tolerance aspects can be modeled. For example, recovery routines can be seen as hard aperiodic tasks which are released upon error detection, when the reconfiguration mechanism can be carried out.

The solution proposed by Jehuda and Israeli [10] also deals with multiple version of tasks. Similarly to our work, their solution aims at maximizing the system benefit subject to the schedulability conditions, which are expressed as processor utilization bounds. The reconfiguration problem can be modeled as the classical knapsack problem, where the processor utilization bound represents the knapsack size. However, this approach may not be suitable to deal with hard aperiodic tasks. Also, the authors have not addressed the problem of mode changes. In fact, in order to carry out system reconfiguration at a given time $t$, one must consider not only the schedulability conditions after the reconfiguration, but also the effects of the old and new configuration that may exist during the interval $[t, t')$ when reconfiguration takes place. Due to this, the work by Jehuda and Israeli is suitable to define meta-policies for reconfiguration and need mode-change mechanisms to be effective. Unlike their approach, we deal with reconfiguration at a lower level, where the system schedulability condition takes into account all the tasks active in the time interval $[t, t')$.

The focus of our work is to provide support to reconfiguration at the scheduler level. Thus, it is possible to ensure timing guaranties and optimize computing resources. Other approaches to adaptation and reconfiguration can be found [1, 9, 17]. However, their goal is to provide a middleware through which applications can negotiate available computing resources. We understand that independently of making use of such middleware infra-structures, dealing with reconfiguration at the scheduler level is necessary and allows for better resource usage.

Rusu *et al.* [18] have proposed a reconfiguration mechanism for energy-aware real-time systems. The goal is to optimize the system benefit subject to both schedulability and energy constraints. Like our solution, they provide reconfiguration at the scheduler level. Two task models are dealt with, the frame-based and the periodic-based task models. The former is more restrictive since all tasks are required to have the same periods and deadlines, which define the frame size. For such a model, the reconfiguration interval $[t, t')$ is the frame size. For the periodic task model the reconfiguration interval is the system hyperperiod, which may involve too many jobs. Unlike their approach, we take into consideration much shorter reconfiguration intervals and a more flexible task model although we do not deal with energy constraints.

## 3 System Model

We consider a uniprocessor system composed of a set of multiversion tasks scheduled by EDF [14]. Tasks do not share resources nor have precedence constraints. Each task generates one or more jobs during the system execution. Jobs are uniquely identified, that is, $J_i$ and $J_j$ are distinct jobs of tasks iff $i \neq j$. The release instant of $J_i$ is denoted $r_i$. If $J_i$ and $J_j$ are released at $r_i$ and $r_j$, then they have to be executed within the time intervals $[r_i, d_i)$ and $[r_j, d_j)$, respectively, where $d_i = r_i + D_i$ and $d_j = r_j + D_j$. The term $D_i$ represents the relative *deadline* of $J_i$ while $d_i$ is its absolute *deadline*. If $J_i$ and $J_j$ are jobs of the same task, $D_i = D_j$. The reconfiguration mechanism described in this paper takes the absolute *deadlines* of jobs, hereafter simply called *deadlines*. Tasks can be periodic, sporadic or aperiodic. There is a fixed and known inter-release time of consecutive jobs of periodic tasks. For sporadic tasks, only their minimum inter-release times are known while aperiodic tasks may be released at any instant.

There is a non-empty set of versions associated with each task. In other words, $J_i$ can be released as one of its $\kappa(i) > 0$ versions. Each version $k$ of $J_i$, $1 \leq k \leq \kappa(i)$, has a worst-case execution time, $C_{ik} \geq 0$. Without loss of generality, we assume that $C_{i1} \geq C_{i2} \geq \ldots \geq C_{i\kappa(i)} \geq 0$. If $J_i$ and $J_j$ are jobs of the same task, $\kappa(i) = \kappa(j)$ and $C_{ik} = C_{jk}$, $k = 1, \ldots, \kappa(i)$. We consider that there is a benefit associated with the execution of the version $k$ of each job $J_i$, denoted $A_{ik}$. The cancellation of $J_i$ is denoted by selecting its version $k$, where $C_{ik} = 0$. It is interesting to notice that temporal protection approaches, based on simply job cancellations, can be modeled as a special case of the proposed reconfiguration model. Indeed, this can be implemented by letting each task have two versions one of which has null cost.

The set $\Gamma(t, t') = \{J_1, \ldots, J_n\}$ represents the jobs active in the interval $[t, t')$. A job $J_i$ is said to be active in this interval if it was released at $r_i < t'$ and has not yet finished its execution by time $t$. Note that there may exist more than a job of the same task in $\Gamma(t, t')$. We define the *interest instant* of $J_i$ as $s_i = \max(t, r_i)$, that is, $s_i$ is the time from which the execution of $J_i$ is considered by the reconfiguration mechanism. Without loss of generality, we assume that $\Gamma(t, t')$ is ordered such that $\forall i < n : d_i < d_{i+1}$ or $d_i = d_{i+1}$ and $s_i \leq s_{i+1}$.

We assume that time is represented as non-negative integers and that the system keeps track of the processing time of each active job $J_i$. Note that this assumption is in line with modern operating systems. Further, we define:

**Definition 1.** *Let $c_{ik}(t)$ be the time already executed as for version $k$ of $J_i$ up to time instant $t$, $0 \leq c_{ik}(t) \leq C_{ik}$. The maximum effective execution time of version $k$ of $J_i$ is*

$$C_{ik}(t) = \begin{cases} C_{ik} - c_{ik}(t) & \textit{if } J_i \in \Gamma(t, t) \\ 0 & \textit{otherwise} \end{cases} \tag{1}$$

Note that if a version $k$ of $J_i$ finished its execution by $t$, $C_{ik}(t) = 0$ and for all other version $l$ of $J_i$ not selected for execution $(l \neq k)$, $C_{il}(t) = C_{il}$, since $c_{il}(t) = 0$.

We assume that the system is schedulable in normal operating conditions. However, there may be overload scenarios during some intervals $[t, t')$. The reconfiguration mechanism must select the version of each active job in $\Gamma(t, t')$ that maximizes the benefit for the system. Overload scenarios can be caused by, say, aperiodic tasks that have to be executed to keep the system correct or in a consistent state. Note that this characteristic prevents the use of mechanisms that deal with soft tasks like aperiodic servers [15, 5]. Also, note that the reconfiguration mechanism affects only jobs in $\Gamma(t, t')$. Jobs that are not active in $[t, t')$ are allowed to execute without degradation.

The reconfiguration mechanism assigns values to the variable $x_{ik} \in \{0, 1\}$ associated with each job in $\Gamma(t, t')$. If $x_{ik} = 1$, then the version $k$ of $J_i$ was selected for scheduling. If $x_{ik} = 1$ and $C_{ik} = 0$, then $J_i$ is canceled or not considered for execution during $[t, t')$. Obviously, there may be jobs that cannot be canceled and this can be modeled by letting $C_{i\kappa(i)} > 0$. As the reconfiguration mechanism must choose exactly one version of each job $J_i$, $\sum_{k \in K_i} x_{ik} = 1$ for all $J_i \in \Gamma(t, t')$, where $K_i = \{1, \ldots, \kappa(i)\}$. Also, there would be a task with one job in $\Gamma(t, t')$, which is responsible for carrying out the system reconfiguration. The system designer must define its execution cost $C$ based on the characteristics of the optimization algorithm used (see next section). For the sake of simplicity, hereafter we do not represent such a job in $\Gamma(t, t')$. This is similar to considering the reconfiguration mechanism starting its execution at $t - C$.

We do not restrict how the benefit values $A_{ik}$ are assigned in the system. Interesting discussion on definitions of benefit functions can be found elsewhere [4] and is beyond the scope of this paper.

## 4 The Reconfiguration Mechanism

The reconfiguration problem addressed in this paper can be stated as follows:

$$P : f = \text{Maximize} \sum_{J_i \in \Gamma(t,t')} \sum_{k \in K_i} A_{ik} x_{ik} \tag{2a}$$

subject to :

$$\text{Schedulability condition} \tag{2b}$$

$$\sum_{k \in K_i} x_{ik} = 1, \forall J_i \in \Gamma(t, t') \tag{2c}$$

$$x_{ik} \in \{0, 1\}, \forall k \in K_i, \forall J_i \in \Gamma(t, t') \tag{2d}$$

Equations (2c) and (2d) mean that the reconfiguration mechanism can only choose one version of each job in $\Gamma(t, t')$. Condition (2b) gives a sufficient schedulability test for the jobs in $\Gamma(t, t')$ and will be derived shortly.

As can be noted, solutions to $P$ may have high computational costs. However, as will be seen, approximation solutions can be derived so that the cost of the dynamic reconfiguration mechanism can be significantly reduced. Another aspect related to the problem complexity is the time interval $[t, t')$ since it determines the input size of the problem. If this interval is too long, reconfiguration may take too long. If it is too short, no feasible solution may be found. In order to determine the reconfiguration interval, we assume that time $t$ is related to the release of some job $J_j$ of an aperiodic task. For example, considering that the reconfiguration mechanism takes $C$ time units to finish and starts at time $t''$, $t = t'' + C$. Time $t'$ can be the release time $r_i$ of the first job $J_i$ that is released at or after $d_j$. This approach bounds the number of jobs considered for reconfiguration and isolates the reconfiguration effects within $[t, t')$. Better approaches to choosing the reconfiguration interval may be possible but such a derivation is beyond the paper scope, which is focused on the reconfiguration procedure itself. Before explaining the solution to problem $P$, we present a simple example in Section 4.1. This example will be used throughout this section for illustration purposes. Then we derive condition (2b) in Section 4.2. An optimal solution to the reconfiguration problem $P$ is given in Section 4.3 and in Section 4.4 a faster and adjustable algorithm for solving $P$ is derived. The evaluation of the proposed solutions is presented in Section 4.5.

### 4.1 Illustrative Example

Consider a reconfiguration interval $[t, t')$, where $t = 0$ and $t' = 234$. During this interval, there are three active jobs, i.e. $\Gamma(0, 234) = \{J_1, J_2, J_3\}$. The release times, deadlines and the versions of these jobs are given in Table 1. As can be seen, jobs $J_1$ and $J_3$ have 10 versions each while $J_2$ has only one. Each job version provides a given quality of service. More specifically, $C_{ik} = C_{i1}(\kappa(i)+1-k)/\kappa(i)$, $i = 1, 3$. For the sake of illustration only, we assume a specific benefit function, $A_{ik} = \frac{C_{ik}}{C_{i1}}$. Note that the system does not allow job cancellation since $C_{i\kappa(i)} > 0$ for all $J_i \in \Gamma(t, t')$.

It is important to emphasize that it is not possible to execute $J_1$, $J_2$ and $J_3$ considering their highest quality versions without missing deadlines. Indeed, the EDF scheduler would select $J_1$ with 31 time units and the remainder time, $101 - 31 = 70$, is not enough to execute $J_2$. The reconfiguration mechanism must select the versions of $J_1$ and $J_3$ so that the system benefit is maximized and no job misses its deadlines. A naive solution would test $10^2$ possible configurations, which is too expensive to be practical.

### 4.2 Schedulability Condition

The focus of this section is on the conditions under which the jobs in $\Gamma(t, t')$ are schedulable in the interval $[t, t')$. By schedulable in the interval we mean that no job $J_i \in \Gamma(t, t')$ finishes its execution after $\min(d_i, t')$.

We define $\xi_i = \sum_{k \in K_i} C_{ik}(s_i) x_{ik}$, i.e. $\xi_i$ is the computation time necessary for the version $k$ of $J_i$ chosen by the reconfiguration mechanism. Also, define $t_i$, $i = 1, 2 \ldots, n$ as follows:

$$t_i = \begin{cases} t' & \text{if } i = n \\ \min(t_{i+1}, d_{i+1}) - \xi_{i+1} & \text{if } 1 \leq i < n \end{cases} \tag{3}$$

|  |  |  |  | $C_{ik}$ |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $J_i$ | $r_i$ | $s_i$ | $d_i$ | $k=1$ | $k=2$ | $k=3$ | $k=4$ | $k=5$ | $k=6$ | $k=7$ | $k=8$ | $k=9$ | $k=10$ |
| $J_1$ | 0 | 0 | 90 | 31.0 | 27.9 | 24.8 | 21.7 | 18.6 | 15.5 | 12.4 | 9.3 | 6.2 | 3.1 |
| $J_2$ | 0 | 0 | 101 | 91.0 | – | – | – | – | – | – | – | – | – |
| $J_3$ | 54 | 54 | 234 | 22.0 | 19.8 | 17.6 | 15.4 | 13.2 | 11.0 | 8.8 | 6.6 | 4.4 | 2.2 |

**Table 1. Jobs in $\Gamma(0, 234)$ and the worst-case execution times of their versions.**

The schedulability condition derived in this section is based on sequencing the jobs in $\Gamma(t, t')$ in the interval $[t, t')$ one after another as if there was no preemption. For example, $J_n$ must be scheduled not before $s_n$ and must finish by $\min(t', d_n)$. In turn, $J_{n-1}$ should start executing not before $s_{n-1}$ but must finish by $\min(t_{n-1}, d_{n-1})$. Since the definition of $t_i$ for $i < n$ takes into consideration the time to execute $J_{i+1}$, i.e. $\xi_{i+1}$, the schedulability of all jobs in $\Gamma(t, t')$ is ensured, as the following theorem states.

**Theorem 1.** *The jobs in* $\Gamma(t, t') = \{J_1, J_2, \ldots, J_n\}$ *are schedulable in the interval* $[t, t')$ *if*

$$\forall J_i \in \Gamma(t, t') : s_i + \xi_i \leq \min(d_i, t_i)\,, \tag{4}$$

*Proof.* The proof will be by constructing a non-preemptive scheduling during $[t, t')$. Consider job $J_n$ first. It is clear that $J_n$ is scheduled if $s_n + \xi_n \leq \min(d_n, t_n)$ since by equation (3), higher priority jobs must have finished their execution by time $t_{n-1} = \min(t_n, d_n) - \xi_n$. In other words, there is a time interval $I_n = [\min(d_n, t_n) - \xi_n, \min(d_n, t_n))$ reserved for the exclusive execution of $J_n$. Now remove $J_n$ from the set of jobs $\Gamma(t, t')$ and let $t' = t_{n-1}$. The same reasoning applies for $\{J_1, J_2, \ldots, J_{n-1}\}$, which leads to the definition of time intervals $I_i = [\min(d_i, t_i) - \xi_i, \min(d_i, t_i))$ that are reserved for execution of $J_i$, $i = 1, 2, \ldots, n$, as if there was no preemption. As $I_i$ is enough to execute $J_i$, schedulability holds, as required. □

It is important to emphasize that we are not considering non-preemptive systems. Non-preemption is assumed only for deriving a schedulability condition. The system may follow the usual EDF ordering and preemptions are allowed. Obviously, the schedulability condition derived is restrictive and so it is only sufficient but can be used to solve the reconfiguration problem in an effective way.

## 4.3 An Optimal Solution

In this section we will give a recursive formulation for reconfiguration problem $P$ based on which we will present an optimal solution. By optimal we mean that the reconfiguration mechanism selects the version $k$ of each job $J_i \in \Gamma(t, t')$ that gives the maximum benefit for the system subject to the schedulability condition (4). This implies that a solution for $P$ is also a solution to the schedulability problem stated by Theorem 1.

Let $P_i(t^*)$ be a restricted version of the reconfiguration problem $P$ which takes into account the job set $\mathcal{J}_i = \{J_1, \ldots, J_i\}$ such that $\mathcal{J}_i$ must be scheduled in the time interval $[t, t^*)$. The goal here is to decompose $P$ into a sequence of subproblems $\langle P_1(t^*), P_2(t^*), \ldots, P_n(t^*)\rangle$, each of which is defined by:

$$P_i(t^*): \; f_i(t^*) = \text{Maximize} \sum_{J_j \in \mathcal{J}_i} \sum_{k \in K_j} A_{jk} x_{jk} \tag{5a}$$

Subject to :

$$s_j + \xi_j \leq \min(d_j, t_j), \tag{5b}$$

$$s_j = \max(t, r_j), \tag{5c}$$

$$\xi_j = \sum_{k \in K_j} C_{jk}(s_j) x_{jk}, \tag{5d}$$

$$\sum_{k \in K_j} x_{jk} = 1, \quad \forall J_j \in \mathcal{J}_i \tag{5e}$$

$$t_j \leq \min(d_{j+1}, t_{j+1}) - \xi_{j+1}, \; \forall J_j \in \mathcal{J}_i - \{J_i\} \tag{5f}$$

$$t_i \leq t^* \tag{5g}$$

$$x_{jk} \in \{0, 1\}, \; \forall k \in K_j, \; \forall J_j \in \mathcal{J}_i \tag{5h}$$

Obviously, $P \equiv P_n(t')$ and so $f = f_n(t')$. In order to derive a recursive formulation for $P_n(t')$, let us consider first the set $\mathcal{J}_1$, which can be expressed as:

$$P_1(t^*): \; f_1(t^*) = \text{Maximize} \sum_{k \in K_1} A_{1k} x_{1k} \tag{6a}$$

Subject to :

$$s_1 + \xi_1 \leq \min(d_1, t_1) \tag{6b}$$

$$s_1 = \max(t, r_1) \tag{6c}$$

$$\xi_1 = \sum_{k \in K_1} C_{1k}(s_1) x_{1k} \tag{6d}$$

$$\sum_{k \in K_1} x_{1k} = 1 \tag{6e}$$

$$t_1 \leq t^* \tag{6f}$$

$$x_{1k} \in \{0, 1\}, \; k \in K_1 \tag{6g}$$

Equation (6d) represents the execution cost of $J_1$ whose version is chosen by the reconfiguration mechanism. It is not difficult to see that the longer $t^*$, the higher the benefit for executing $J_1$. It is clear that the bound on this benefit must take into consideration the jobs $J_i \in \Gamma(t,t')$, $i > 1$. In other words, to solve problem $P_1(t^*)$, one has to consider the time interval $[t, t^*)$ taking into account time $\xi_2$, which is necessary for executing $J_2$. In turn, the scheduling of $J_2$ cannot be done without observing $J_3$, whose execution cost is $\xi_3$ and so on. This leads to the following recursive formulation for the subproblems $P_i(t^*)$, $i > 1$:

$$P_i(t^*):\ f_i(t^*) = \text{Maximize} \sum_{k\in K_i} A_{ik}x_{ik} + f_{i-1}(\min(d_i,t_i) - \xi_i) \tag{7a}$$

Subject to :

$$s_i + \xi_i \le \min(d_i, t_i) \tag{7b}$$

$$s_i = \max(t, r_i) \tag{7c}$$

$$\xi_i = \sum_{k\in K_i} C_{ik}(s_i)x_{ik} \tag{7d}$$

$$\sum_{k\in K_i} x_{ik} = 1 \tag{7e}$$

$$t_i \le t^* \tag{7f}$$

$$x_{ik} \in \{0,1\},\ k \in K_i \tag{7g}$$

The formulation for $P_1(t^*)$ and $P_i(t^*)$ $(i > 1)$ can be expressed in a more compact form. Let us consider two scenarios, depending on whether or not a version $k$ of $J_i$ can be scheduled in the time interval $[t, t^*)$, which can be verified by condition $s_i + C_{i\kappa(i)}(s_i) \le \min(d_i, t^*)$:

**Case 1**. No version $k$ of $J_i$ is schedulable in the interval. This case may take place when $s_i + C_{i\kappa(i)}(s_i) > \min(d_i, t^*)$. There are two subcases:

**Case 1a**. Job $J_i$ can be canceled, i.e. $C_{i\kappa(i)}(s_i) = 0$. Thus, no execution cost nor benefit related to $J_i$ will be considered. This means that $f_i(t^*) = 0$ if $i = 1$ or $f_i(t^*) = f_{i-1}(t^*)$ for all $i > 1$.

**Case 1b**. The cancellation of $J_i$ is not possible, which is indicated by $C_{i\kappa(1)}(s_i) > 0$. Thus, no feasible solution can be found. This is represented by letting $f_i(t^*) = -\infty$.

**Case 2**. Some version $k$ of $J_i$ is schedulable, that is, $\exists k : s_i + C_{ik}(s_i) \le \min(d_i, t^*)$. In this case, $f_i(t^*)$ assumes the maximum benefit $A_{ik}$ for all schedulable versions $k$. If $i > 1$, it is needed to add $f_{i-1}(\min(d_i, t^*) - C_{ik}(s_i))$ to $A_{ik}$ and so $f_i(t^*) = \max_{k\in K_i}\{A_{ik} + f_{i-1}(\min(d_i, t^*) - C_{ik}(s_i) : s_i + C_{ik}(s_i) \le \min(d_i, t^*)\}$. Note that the time interval considered for executing $J_{i-1}$ is reduced by the time spent by the execution of $J_i$.

The formulation expressed by equations (7a) to (7g) leads to a dynamic programming algorithm [22], which is given below. Lines 1-13 of the algorithm refer to $f_1(t^*)$ for $t^* = t, t+1, \ldots, t'$. The values of $f_i(t^*)$, $i > 1$, are computed in lines 14-28 for the same interval for $t^*$. Table $p_i(t^*)$ stores the value of $k \in K_i$ that leads to the maximum benefit for each $t^*$.

```
DP-Solve-P$(t, t', n, \kappa, A, C, r, d)$
1:  $s_1 \leftarrow \max(t, r_1)$
2:  for $t^* = t$ to $t'$ do
3:      if $C_{1\kappa(1)} > 0 \wedge s_1 + C_{1\kappa(1)}(s_1) > \min(d_1, t^*)$ then
4:          $p_1(t^*) \leftarrow 0$
5:          $f_1(t^*) \leftarrow -\infty$
6:      else if $C_{1\kappa(1)} = 0 \wedge s_1 > \min(d_1, t^*)$ then
7:          $p_1(t^*) \leftarrow \kappa(1)$
8:          $f_1(t^*) \leftarrow 0$
9:      else
10:         $p_1(t^*) \leftarrow \arg\max_{k\in K_1}\{A_{1k} : s_1 + C_{1k}(s_1) \le \min(d_1, t^*)\}$
11:         $f_1(t^*) \leftarrow A_{1p_1(t^*)}$
12:     end if
13: end for
14: for $i = 2$ to $n$ do
15:     $s_i \leftarrow \max(t, r_i)$
16:     for $t^* = t$ to $t'$ do
17:         if $C_{i\kappa(i)} > 0 \wedge s_i + C_{i\kappa(i)}(s_i) > \min(d_i, t^*)$ then
18:             $p_i(t^*) \leftarrow 0$
19:             $f_i(t^*) \leftarrow -\infty$
20:         else if $C_{i\kappa(i)} = 0 \wedge s_i > \min(d_i, t^*)$ then
21:             $p_i(t^*) \leftarrow \kappa(i)$
22:             $f_i(t^*) \leftarrow f_{i-1}(t^*)$
23:         else
24:             $p_i(t^*) \leftarrow \arg\max_{k\in K_i}\{A_{ik} + f_{i-1}(\min(d_i, t^*) - C_{ik}(s_i)) : s_i + C_{ik}(s_i) \le \min(d_i, t^*)\}$
25:             $f_i(t^*) \leftarrow A_{ip_i(t^*)} + f_{i-1}(\min(d_i, t^*) - C_{ip_i(t^*)}(s_i))$
26:         end if
27:     end for
28: end for
29: return $(p, f)$
```

The execution time of the algorithm is $O((t'-t)n\kappa_{max})$, where $\kappa_{max} = \max\{\kappa(i) : J_i \in \Gamma(t,t')\}$. Thus, it is pseudo-polynomial in the size of the input. The memory complexity is $\Theta(n(t'-t))$ for storing $f_i(t^*)$ and $p_i(t^*)$.

It can be noticed that the reconfiguration problem has the same structure of the standard knapsack problem with processor time corresponding to the knapsack capacity. However, the former generalizes the latter in two ways. First, the weight of each item (execution time) can be selected from a set of discrete values. Second, each item should fit

|  | $f_i(t^*)$ |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| $J_i \backslash t^*$ | 0 | 10 | 20 | 60 | 120 | 212 | 234 |
| $J_1$ | $-\infty$ | 0.3 | 0.6 | 1.0 | 1.0 | 1.0 | 1.0 |
| $J_2$ | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ | 1.3 | 1.3 | 1.3 |
| $J_3$ | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ | 2.2 | 2.3 | 2.3 |

**Table 2. $f_i(t^*)$ for the illustrative example.**

|  | $p_i(t^*)$ |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
| $J_i \backslash t^*$ | 0 | 10 | 20 | 60 | 100 | 120 | 212 | 234 |
| $J_1$ | 0 | 8 | 5 | 1 | 1 | 1 | 1 | 1 |
| $J_2$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| $J_3$ | 0 | 0 | 0 | 0 | 9 | 1 | 1 | 1 |

**Table 3. $p_i(t^*)$ for the illustrative example.**

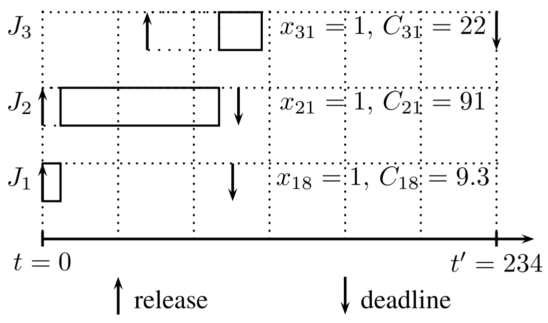
in a given part of the knapsack, meaning the time extending from the release time to the deadline. It is then not surprising that a dynamic programming algorithm was designed for the reconfiguration problem. However, an efficient algorithm was not found for the dual form of the recursive formulation, which would lead to a fully-polynomial approximation scheme.

Applying algorithm **DP-Solve-P** to the illustrative example given in Section 4.1 leads to Tables 2 and 3, which contain the values for $f_i(t^*)$ and $p_i(t^*)$, respectively. For the sake of illustration, only some values are shown in the tables. Since $p_3(234) = 1$, which implies that $x_{3,1} = 1$, the highest quality version of $J_3$ is selected for execution. Thus, the remainder time interval $\min(234, d_3) - C_{3,1} = 234 - 22 = 212$ is considered for executing the other two jobs. The benefit regarding $J_3$ is $A_{3,1} = 1$. Then subproblem $P_2(212)$ is to be solved. As $p_2(212) = 1$, $x_{2,1} = 1$ and the highest quality version of $J_2$ is set to start executing at $\min(212, d_2) - C_{2,1} = 101 - 91 = 10$ offering a benefit $A_{2,1} = 1$. Then the subproblem $P_1(10)$ needs to be solved. As $p_1(10) = 8$ and $x_{1,8} = 1$, $J_1$ is scheduled at time $\lfloor \min(10, d_1) - C_{1,8} \rfloor = \lfloor 10 - 9,3 \rfloor = \lfloor 0,7 \rfloor = 0$, adding up a benefit $A_{1,8} = 0.3$. The solution provided by algorithm **DP-Solve-P** is the same as the one obtained by solving problem $P$, as indicated in Table 4. The scheduling representation of the solution is presented in Figure 1.

Since we are using dynamic programming, all the values of $C_{ik}$ must be integer parameters. In practice, when non-integer values are present, one needs to scale the timing values by multiplying them by a sufficient large constant so that $C_{ik}$ belongs to the integer domain. In the case of the illustrative example, all the values of $C_{ik}$ and the time instants $t$ and $t'$ were multiplied by 10.

## 4.4 Approximation Solution

Algorithm **DP-Solve-P** offers a basis for comparison since it provides the optimal solution subject to the condition stated in Theorem 1. If the system has a low num-



$J_3$
$x_{31} = 1,\ C_{31} = 22$
$J_2$
$x_{21} = 1,\ C_{21} = 91$
$J_1$
$x_{18} = 1,\ C_{18} = 9.3$
$t = 0$
$t' = 234$
release
deadline

**Figure 1. Solution for the illustrative example.**

| $J_i$ | $\xi_i$ | $t_i$ | $\min(d_i, t_i)$ | $x_{i,k}$ |
|---|---|---|---|---|
| $J_1$ | 9.3 | 9.3 | 9.3 | $x_{1,8} = 1$ |
| $J_2$ | 91.0 | 212.0 | 101.0 | $x_{2,1} = 1$ |
| $J_3$ | 22.0 | 234.0 | 234.0 | $x_{3,1} = 1$ |

**Table 4. Solution for the illustrative example.**

ber of jobs or job versions in $\Gamma(t, t')$, applying directly this optimal solution may be viable. However, its pseudo-polynomial execution time may prevent its on-line use for larger systems. In this section we derive an approximation solution to problem $P$, which gives lower bounds on the optimal solutions found by algorithm **DP-Solve-P** and significantly reduces its time and memory complexities. The main idea of the approximation [21] is to reduce the execution times $C_{ik}(s_i)$ of the input by dividing them by a constant $\alpha > 1$.

Let $b$, $c_j$ be constants and $x_j \in \{0, 1\}$ be variables, $j = 1, \ldots, n$ and consider the following relation:

$$\sum_{j=1}^{n} c_j x_j \le b\,, \tag{8}$$

Suppose that the problem is to find an assignment $\mathbf{x}$ for $x_j$ $(j = 1, \ldots, n)$ such that relation (8) holds. First, consider the following modified relation:

$$\sum_{j=1}^{n} \lceil c_j \rceil x_j = \sum_{j=1}^{n} \lceil c_j x_j \rceil \le \lfloor b \rfloor \Rightarrow \lceil \sum_{j=1}^{n} c_j x_j \rceil \le \lfloor b \rfloor \tag{9}$$

It is clear that a solution that satisfies relation (9) also satisfies relation (8) (the opposite is not necessarily true). The relation (9) allows us to generate an approximation model $P_\alpha$ for $P$, as a function of a constant $\alpha \ge 1$, by substituting:

1. $C_{ik}(s_i)$ by $\lceil C_{ik}(s_i)/\alpha \rceil$, $\forall J_i \in \Gamma(t, t'),\ k \in K_i$;
2. $r_i$ by $\lceil r_i/\alpha \rceil$ and $d_i$ by $\lfloor d_i/\alpha \rfloor$, $\forall J_i \in \Gamma(t, t')$; and
3. $t'$ by $\lfloor t'/\alpha \rfloor$ and $t$ by $\lceil t/\alpha \rceil$.

Observe that any feasible solution $\mathbf{x}$ for $P_\alpha$ is feasible for $P$ and so $P_\alpha$ can be used as an approximation of $P$. Moreover, the value of the objective function $f(\mathbf{x})$ defines a lower bound for the optimal solution of $f$. On the other hand, the execution time of the dynamic programming algorithm applied to $P_\alpha$ is reduced to $O((t' - t)n\kappa_{max}/\alpha)$.

Further, using similar reasoning one may derive upper bounds on the objective function. Indeed, it follows that:

$$\sum_{j=1}^{n} c_j x_j \leq b \Rightarrow \sum_{j=1}^{n} \lfloor c_j \rfloor x_j \leq \lceil b \rceil \tag{10}$$

Thus, it is possible to derive a relaxed relation $\bar{P}_\alpha$ of $P$ for a constant $\alpha \geq 1$ by substituting:

1. $C_{ik}(s_i)$ by $\lfloor C_{ik}(s_i)/\alpha \rfloor$, $\forall J_i \in \Gamma(t, t')$, $k \in K_i$;
2. $r_i$ by $\lfloor r_i/\alpha \rfloor$ and $d_i$ by $\lceil d_i/\alpha \rceil$, $\forall J_i \in \Gamma(t, t')$; and
3. $t'$ by $\lceil t'/\alpha \rceil$ and $t$ by $\lfloor t/\alpha \rfloor$.

A solution for $P$ is also a solution for $\bar{P}_\alpha$, but the opposite does not necessarily hold. As both problems have the same objective function, it follows that $\bar{P}_\alpha$ is a relaxed version of $P$. This implies that $\bar{P}_\alpha$ cannot be used as an approximation of the reconfiguration problem $P$. Our intention here is only to determine an upper bound for $P$. A solution of $\bar{P}_\alpha$ can be obtained by the derived dynamic programming algorithm, but now in time $O((t' - t)n\kappa_{max}/\alpha)$.

| | Rounding factor $\alpha$ | | | | |
|---|---|---|---|---|---|
| | 1 | 8 | 16 | 32 | 64 |
| Runtime | 70,200 | 17,550 | 273 | 69 | 16 |
| Upper bound | 2.3 | 2.3 | 2.3 | 2.4 | 2.6 |
| Lower bound | 2.3 | 2.3 | 2.3 | 2.3 | $-\infty$ |

**Table 5. Reconfiguration runtime and the upper and lower bounds on the benefit found for the illustrative example.**

Table 5 shows data relative to the reconfiguration procedure applied to the illustrative example. As can be seen from the table, the algorithm speeds up considerably as $\alpha$ increases. The runtime is given in time units. The same time unit value was used to represent time in Table 1. Also, note that the lower and upper bounds give the same value as or are very close to the optimal benefit.

It is important to notice that the higher the value of $\alpha$, the lower the accuracy of the solution and in extreme cases the approximation degrades to the point at which no solution can be found, which is the case for $\alpha \geq 64$, as shown in the table. This side-effect takes place because the timing parameters may be reduced too much after dividing them by $\alpha$. For example, suppose that $C_{ik}/\alpha < 1$ $(\forall k \in K_i)$. Thus, the execution costs of all versions of job $J_i$ are made equal to 0 or 1 depending on which rounding approach is carried out. Also, it is clear that this rounding effect may increase the pessimism of the schedulability test. These side-effects must be considered by the system designer when choosing a suitable value for $\alpha$. The simulation results will better illustrate such an aspect.

One may notice that the time spent to find out a solution to the reconfiguration problem cannot be neglected. Nevertheless, since without running an optimization algorithm one is not able to find a feasible solution that maximizes the system benefit, the value of $\alpha$ must be chosen taking into consideration the time and memory available to run the reconfiguration procedure and the level of approximation allowed.

## 4.5 Simulation Results

Extensive simulation to evaluate the proposed reconfiguration solution was carried out. This section presents some of the relevant results found during the simulation. Task sets with 8 periodic tasks each were randomly generated as follows. For each periodic task, the worst-case execution task of its highest quality version was generated according to an exponential distribution with parameter $u_p/10$, where $u_p = 40\%, 50\%, \ldots, 90\%$ is the processor load (utilization) for periodic tasks. All periodic tasks had 10 versions each and their worst-case execution times were set to $C_{i,k+1} = 90\% C_{ik}$ $(k = 1, 2, \ldots, 9)$. This means that periodic tasks could not be cancelled. Task periods were generated according to a uniform distribution in the interval 80 to 500 time units and their deadlines were considered to be equal to their periods. The simulation time was defined as $100,000$ time units and the first jobs of all periodic tasks were assumed to be released at the beginning of the simulation.

During the simulation, the jobs of aperiodic tasks were generated according to a Poisson distribution with parameter $1/100$, which means that there were on average $1,000$ jobs of aperiodic tasks during each simulation run. Each of these jobs determines a reconfiguration interval $[t, t')$. Instant $t$ corresponds to the release time of the job while $t'$ is computed as explained in Section 4. These jobs had two versions. The worst-case execution times of their first versions were generated according to an exponential distribution with parameter $0.2$. Their second versions had null computation cost, which means that they could be rejected by the reconfiguration mechanism. The deadline of each generated aperiodic job $J_i$ was defined as $d_i = t + C_{i1}/u_a$, where $u_a$ is the computing demand of $J_i$ within $[t, d_i)$. Several values of $u_a$ were considered during the simulation. Here we present the results for $u_a = 40\%$ since it illustrates well the behavior of the reconfiguration mechanism. Similar behaviors were found for other values of $u_a$.

Figures 2 and 3 plot the results found during the simulation and illustrate the behavior of the approximation approach for different values of $\alpha$. The data shown in the graphs are the average values found for each reconfiguration carried out during the simulation. As can be seen from Figure 2, the number of aperiodic jobs rejected increases considerably when $\alpha$ increases. This is expected since, as mentioned in Section 4.4, higher values of $\alpha$ introduces more pessimism in the schedulability test. Also, job rejection for the same value $\alpha$ tends to increase slightly when $u_p$ increases. This is because the reconfiguration procedure chooses to gracefully degrade the jobs of periodic tasks rather than rejecting aperiodic jobs, which was expected by the defined benefit functions (see this definition below). Indeed, when accepting aperiodic jobs, the higher the periodic load, the higher the degradation level and this tends to keep the effective periodic load constant. When an aperiodic job is rejected it is because no feasible configuration that accepts such a job is found. Slightly higher variation can be observed for $\alpha = 32$. This can be explained by the loss of accuracy due to rounding effects. For illustration purposes only, we plot what would be the rejected aperiodic jobs if no degradation of periodic jobs was possible (solid line in the graph).

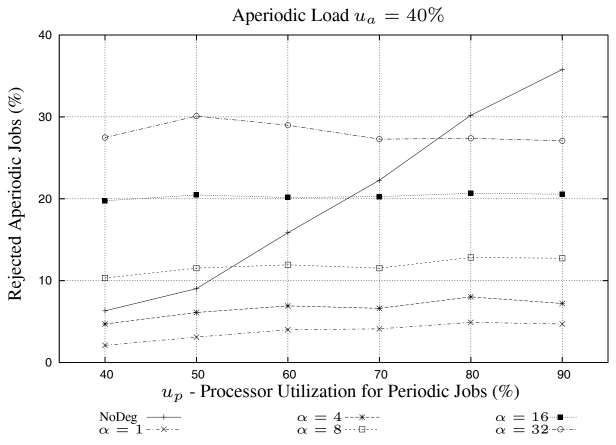


**Figure 2. Rejected aperiodic jobs.**

It is clear that by rejecting aperiodic jobs, the system does not need any reconfiguration. This means that depending on its benefit function, the system benefit could be higher for higher values of $\alpha$ since this may raise the rejection of aperiodic jobs. Thus, to carry out the evaluation, we needed to isolate the effects of jobs rejection when running the approximation approach in order not to compute accuracy loss for higher values of $\alpha$ (due to the rounding effects of the approximation), as if there were better performance in reconfiguring the system. To do this we defined two distinct benefit functions for periodic and aperiodic jobs, respectively. If $J_i$ is periodic, $A_{ik} = \frac{C_{ik}}{C_{i1}}$. Otherwise, $A_{i1} = n$ and $A_{i2} = 0$, where $n$ is the number of periodic jobs in $\Gamma(t, t')$.

The benefit achieved by the reconfiguration procedure is shown in Figure 3. For the sake of illustration, we plotted the benefit when reconfiguration was not allowed, which is indicated by the solid line. The other lines in the graph represent the benefit achieved by the reconfiguration mechanism for different values of $\alpha$. As can be seen, the optimal value is achieved for $\alpha = 1$ and the lowest for $\alpha = 32$, as expected. It is interesting to note that $\alpha \leq 16$ gives benefit values very close to the optimum. Indeed, the biggest difference is at $u_p = 80\%$, where the benefit achieved for $\alpha = 16$ is 7.2% of the optimum.

It is interesting to contrast the benefit achieved by the reconfiguration procedure with its runtime. The time taken by function $(t' - t)n\kappa_{max}/\alpha$ for each reconfiguration interval $[t, t')$ carried out during the simulation runs was measured. The average values for $\alpha = 1, 4, 8, 16, 32$ are $1,571.14$, 407.72, 208.02, 103.18, and 92.12, respectively. As can be noticed, for the kind of system we simulated, $\alpha = 8$ or $\alpha = 16$ can give cost-effective choices for implementing the reconfiguration procedure. For example, with $\alpha = 8$ one can reduce the reconfiguration running time in more than $86\%$ and still achieve the benefit very close to the optimum, which can be considered an excelent result.

It is clear that the system designers must choose the value of $\alpha$ according to the system characteristics. The value of $\alpha$ must be chosen taking into account the execution cost of the reconfiguration task. For that, some evaluation experiments must be carried out. For example, selecting $\alpha = 16$, $103.18$ time units should be reserved to run the reconfiguration procedure.

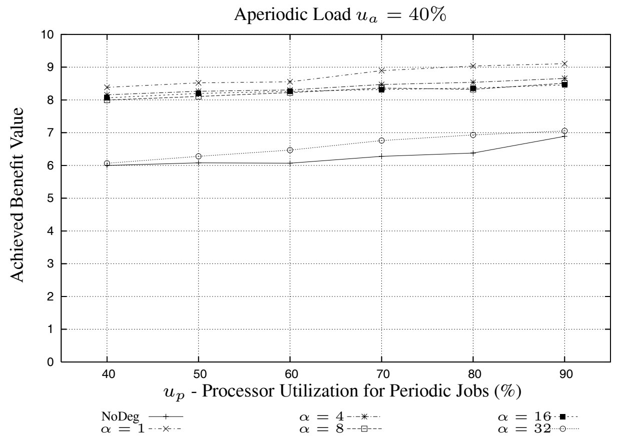


**Figure 3. System benefit.**

## 5 Conclusions

We have presented an approach to dynamic reconfiguration for supporting modern real-time systems structured as a set of multiversion tasks. First, we have derived a sufficient schedulability condition based on which it is possible to check whether a set of jobs is schedulable within a given time interval. Then, based on this condition and using dynamic programming techniques, we have derived an optimal solution for the reconfiguration problem. As this solution is pseudo-polynomial and so it may be considered time and memory consuming for being used on-line, we have also presented an approximation solution to the same problem. Extensive simulation-based evaluation indicates that one can get good trade-offs using the approximation. Indeed, simulation data have shown that runtime can dramatically be reduced at the expense of a small decrease in the system benefit.

Dynamic reconfiguration of a system is a complex problem that requires both efficient schedulability conditions and fast optimization solutions. Deriving less restrictive schedulability conditions that can lead to better reconfiguration mechanisms is a current research topic. Since not all schedulability conditions lead to efficient optimization algorithms, both schedulability and optimization algorithms must be dealt with in conjunction. Moreover, it would be interesting to investigate efficient heuristics that can be used to solve the reconfiguration problem. Although this problem is similar to the knapsack problem, the derivation of such heuristics turned out to be not simple. Another research issue for future work is to incorporate the temporal isolation aspect into the reconfiguration mechanism, possibly using some type of aperiodic server. The problem addressed in this paper and the solution provided here can certainly be used as a basis for such research directions.

## References

[1] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin. “QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control”. *IEEE Transactions on Computers*, 49(11):1170–1183, 2000.

[2] G. Beccari, S. Caselli, and F. Zanichelli. “A Technique for Adaptive Scheduling of Soft Real-Time Tasks”. *Real-Time Systems*, 30(3):187–215, July 2005.

[3] G. Bernat, A. Burns, and A. Llamosi. “Weakly Hard Real-Time Systems”. *IEEE Transactions on Computers*, 50(4):308–321, 2001.

[4] A. Burns, D. Prasad, A. Bondavalli, F. D. Giandomenico, K. Ramamritham, J. Stankovic, and L. Stringini. “The Meaning and Role of Value in Scheduling Flexible Real-Time Systems”. *Journal of Systems Architecture*, 46(4):305–325, 2000.

[5] G. Buttazzo. *“Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications”*. Kluwer Academic Publishers, 1997.

[6] G. Buttazzo and L. Abeni. “Adaptive Workload Management through Elastic Scheduling”. *Real-Time Systems*, 23(1-2):7–24, 2002.

[7] G. Buttazzo and J. Stankovic. “RED: Robust Earliest Deadline Scheduling”. In *Proc. of The Third International Workshop on Responsive Computing Systems*, pages 100–111, 1993.

[8] T. Chantem, X. S. Hu, and M. Lemmon. “Generalized Elastic Scheduling”. In *Proc. of the 27th Real-Time Systems Symposium (RTSS 06)*, pages 236–245, 2006.

[9] D. Hull, A. Shankar, K. Nahrstedt, and J. W. S. Liu. “An End-to-End QoS Model and Management Architecture”. In *Proc. of IEEE Workshop on Middleware for Distributed Real-time Systems and Services*, 1997.

[10] J. Jehuda and A. Israeli. “Automated Meta-Control for Adaptable Real-Time Software”. *Real-Time Systems*, 14(2):107–134, 1998.

[11] G. Koren and D. Shasha. “Skip-Over: Algorithms and Complexity for Overloaded Systems that Allow Skips”. In *Proc. of the 16th IEEE Real-Time Systems Symposium (RTSS ’95)*, pages 110–117, Washington, DC, USA, 1995.

[12] T.-W. Kuo and A. K. Mok. “Incremental Reconfiguration and Load Adjustment in Adaptive Real-Time Systems”. *IEEE Transactions on Computers*, 46(12):1313–1324, 1997.

[13] G. M. A. Lima and A. Burns. “An Optimal Fixed-Priority Assignment Algorithm for Supporting Fault Tolerant Hard Real-Time Systems”. *IEEE Transactions on Computers*, 52(10):1332–1346, 2003.

[14] C. L. Liu and J. W. Layland. “Scheduling Algorithms for Multiprogram in a Hard Real-Time Environment”. *Journal of ACM*, 20(1):40–61, 1973.

[15] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, 2000.

[16] P. Ramanathan and M. Hamdaoui. “A Dynamic Priority Assignment Technique for Streams with (m, k)-Firm Deadlines”. *IEEE Transactions on Computer*, 44(12):1443–1451, 1995.

[17] D. Rosu, K. Schwan, and S. Yalamanchili. “FARA- A Framework for Adaptive Resource Allocation in Complex Real-Time Systems”. In *Proc. of the 4th IEEE Real-Time Technology and Applications Symposium. (RTTAS 1998)*, pages 79–84, 1998.

[18] C. Rusu, R. Melhem, and D. Mossé. “Multi-version Scheduling in Rechargeable Energy-aware Real-Time Systems”. *Journal of Embedded Computing*, 1(2):271–283, 2005.

[19] P. Shriver. “Opening the Door to Smart Power Management in Small Satellites”. In *Proc. of the AIAA/USU Conference on Small Satellites*, 2003.

[20] M. Spuri and G. Buttazzo. “Scheduling Aperiodic Tasks in Dynamic Priority Systems”. *Real-Time Systems*, 10(2):179–210, 1996.

[21] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, 2001.

[22] L. A. Wolsey. *Integer Programming*. John Wiley & Sons, New York, NY, 1998.