Dynamic Reconfiguration in Reservation-based Scheduling: An Optimization Approach*

Augusto Born de Oliveira, Eduardo Camponogara
Depart. of Automation and Systems Engineering (DAS)
Federal University of Santa Catarina (UFSC)
Florianópolis, SC, Brazil
{augustob,camponog}@das.ufsc.br

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

Abstract

Reservation-based scheduling mechanisms have successfully been used for supporting real-time applications whose tasks exhibit high variability in their execution or release times. Indeed, such mechanisms are able to preallocate system bandwidth to the application tasks so that temporal isolation between them is ensured. However, bandwidth allocation is usually based on off-line policies, which may not be suitable for real-time applications that are structured as having several modes of operation, each one requiring a distinct level of system bandwidth. Variations in light conditions, the changing of energy levels, error-detection, or operator commands are examples of events that may trigger a different mode of operation in multi-mode adaptive real-time applications.

In this paper we address the problem of dynamically reconfiguring scheduling parameters of reservation-based mechanisms, offering support for multi-mode adaptive real-time applications. Assuming that each reconfiguration option gives a benefit for the system, reconfiguration is seen as an optimization problem whose objective is to maximize the overall system benefit. Two different models for the problem are formulated, the Integer Programming (IP) and the Linear Programming (LP) formulations. The IP formulation gives rise to an NP-Hard problem for which we give efficient approximate solutions. Also, an optimal and polynomial solution to the LP formulation is derived. Results obtained from extensive simulation indicate the good performance of the proposed reconfiguration mechanisms.

*This research has been supported in part by CNPq under grants 473841/2007-0 and 475851/2006-4, by FAPESB under grant 7320/2007 and by CAPES under grant PROCAD - AST-BAHIA 0271-05-5.

1 Introduction

The application spectrum of real-time systems has broadened considerably in the past few decades. Such systems, once characterized by having simple periodic predictable behavior [15], have become much more complex. Indeed, modern hardware and software architectures make it difficult, or even impossible, to estimate worst-case computation time of real-time services accurately. Furthermore, the environment within which real-time systems are being applied may be highly dynamic.

Reservation-based scheduling has proven to be a suitable choice when dealing with systems that exhibit high variability in the execution or the release times of their tasks [17]. Usually, this approach is implemented by structuring the system as a set of servers $\mathcal{S}$. Each server $S_i \in \mathcal{S}$ is defined in terms of two parameters, (Q_i, T_i), where T_i is the server period and Q_i is its maximum budget, meaning that the tasks scheduled by S_i may use up to $u_i = Q_i/T_i$ of processor. Such an approach provides temporal isolation between system tasks so that time faults (e.g. overruns) of some tasks do not affect the timeliness of the others.

Usually, the server parameters (Q_i, T_i) are statically defined during design time. However, in a system with various applications, there can be events at which processor time needs to be redistributed between the tasks during runtime. Indeed, the need of dynamically adjusting scheduling parameters, due to external sensory data or low-level architecture features has been recently pointed out [5]. In particular, reconfigurable systems can be structured as having different modes of operations. Several examples can be given. In a surveillance system, if it is detected motion in a monitored room, it might be needed to shift computing power to the video recording system to capture better images of a possible intruder. By zooming-in a movie screen in a multimedia player, the user is indicating a different mode of operation, where certain tasks change the relative importance in

the system. The computer vision subsystem of a robot may experience different operational modes due to environment changes. Light conditions, obstacles, vision angle, modifications in the robot goals during its lifetime, and other unpredictable environmental characteristics may be modeled by different operation modes. In this context, support for switching from one operation mode to another is needed at the scheduling level.

In this paper we address the problem of reconfiguring reservation-based scheduling parameters at runtime. More specifically, we assume that different server parameter assignments may yield distinct benefits for the system. Thus, the reconfiguration problem is seen as an optimization problem according to which the overall system benefit should be maximized. The derived techniques are applied to systems based on the Constant Bandwidth Server (CBS) scheduling approach [2]. Two models of the reconfiguration problem are formulated. The first model assumes that there are predefined values of (Q_i, T_i) for each server S_i, which fits into the *Integer Programming* (IP) class of optimization problems. This model is particularly suitable for applications whose tasks have alternative implementations and/or expected inter-release times. For example, different decodification algorithms or decodification rates can be associated to the processing of a movie scene. Higher (lower) execution costs/processing rates give higher (lower) quality of service. As will be seen, the IP formulation implies an NP-hard optimization problem. Two very efficient approximation solutions to this problem are given.

The second model formulates the reconfiguration problem as a *Linear Programming* (LP) problem. The values of (Q_i, T_i) are not assumed to be predefined. The goal is then to find out which bandwidth values u_i maximize the overall system benefit. We show that the optimal solution to this LP problem can be analytically found and its implementation is very efficient.

The solutions to both IP and LP versions of the reconfiguration problem are evaluated by extensive simulation. As will be seen, obtained results show that the approximate solutions for the IP formulation are very close to the optimum even though their execution time is several times shorter. Both formulations are shown to have comparable execution times. The benefits of applying the proposed approach are verified through simulation of a reconfiguration request using a video decoder workload.

This paper is divided into seven sections. After briefly commenting on related work in Section 2, we define the computational model and present some notation in Section 3. The reconfiguration problems addressed in this paper are defined in Section 4 and their solutions are derived in Section 5. Results from extensive simulation to evaluate the described algorithms are presented in Section 6. Conclusions are drawn in Section 7.

2 Related work

Dynamic reconfiguration has been previously addressed by other researches [4, 6, 10, 11, 13, 20] but most of them do not deal with temporal isolation or reservation-based scheduling.

In particular, the approach described in [12] models QoS adaptation for multimedia systems using optimization formulations, where the objective is to maximize the overall system benefit. Although we share the same goal, our approach differs from it in the sense that our focus on scheduling allows for the creation of a reconfiguration infrastructure that accounts for issues such as fairness and mode-changing, and direct integration with existing schedulers.

In the context of CBS, there have been interesting developments in dynamically adjusting the server parameters by using feedback control theory [23, 1, 3, 19, 16]. The main idea of these approaches is to actuate on the server parameters, adjusting their bandwidth, so that a certain QoS metric is as close to the desired value as possible. For example, let the scheduling error of a job be the difference between its finishing time and its server deadline [1]. If the jobs of a task finish too late (too early), more (less) bandwidth should be allocated to its server. We understand that our approach is complementary to feedback scheduling since once a reconfiguration is carried out, feedback scheduling techniques can provide further necessary fine-grained adjustments. The main difficulty in applying these approaches is determining a model for system dynamics so that a stabilizing control law can be designed.

Unlike most of the above approaches, we are interested in providing support for coarse-grained adjustments in server parameters at runtime. This kind of support is necessary when applications switch from one mode of operation to another. This kind of abrupt changing may not be dealt with properly by feedback scheduling techniques. This behavior will be illustrated by some experimental results we provide in this paper.

The reconfiguration problem is formulated here as an optimization problem. We give two formulation versions, modeled as IP and LP optimization problems. Preliminary results regarding the IP formulation appeared in [9]. Also, this version has recently been solved using a Genetic Algorithm technique [21]. However, due to its slow convergence time, this solution is more appropriate to define metaheuristics and must not be used when the time to reconfigure the system is an issue. Here, we extend the former solutions regarding the IP formulation, showing two approximation algorithms, which are thoroughly evaluated by simulation. Furthermore, an LP formulation is presented and its optimal solution is analytically derived. Both solutions are compared via simulation.

3 System Model and Notation

We consider a system with a single processor. The system tasks are scheduled by a reservation-based scheme. In particular, we assume that there are n Constant Bandwidth Servers (CBS) $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ [2]. Although other reservation-based schemes are in line with the solutions presented in this paper, we chose CBS because it is a well known reservation-based scheduling and it is the basis for several other approaches [8, 7, 14].

According to the CBS rules, each server $S_i \in \mathcal{S}$ is defined by the tuple (Q_i, T_i), where Q_i defines its maximum budget and T_i represents its period. Each server S_i's utilization is at most $u_i = Q_i/T_i$ and serves a specific set of tasks. In other words, a system constructed in this manner allocates, for each set of tasks attributed to server S_i, a constant bandwidth determined by u_i, providing temporal isolation between servers even in overload conditions. The budget of each server S_i is depleted as its jobs are executed. To guarantee that the server obeys its utilization limit, the deadline of S_i is postponed by T_i every time its budget is depleted. At the same time, full budget Q_i is restored. When a new job arrives in an idle CBS, budget restoration and deadline postponement depend on a condition calculated at runtime. For more details refer to [2]. Usually, the values of the parameters (Q_i, T_i) are statically determined. The role of the reconfiguration mechanisms described herein is to choose suitable values for each server parameters at runtime.

We assume that there is no precedence relation nor shared resources between tasks. Thus, since every server is scheduled according to EDF, 100% utilization can be attained when scheduling the servers in $\mathcal{S}$. That is, the system is schedulable if and only if [1]

$$\sum_{S_i \in \mathcal{S}} u_i \leq 1 \tag{1}$$

In the context of this work, we assume that the system/user can, at any time, request a reconfiguration of the parameters of the servers in $\mathcal{S}$. This kind of request is associated to changing the operation mode of the system. To accomplish this, the application indicates what processor time percentages U_i should be allocated to each server S_i and a benefit value v_i associated to such an allocation. The benefit value actually achieved depends on the bandwidth u_i the system is able to set. For example, by selecting a movie screen whose processing is being served by S_i, the user may be indicating that U_i should be given to S_i so that the benefit v_i can be attained. However, if the system is able to allocate $u_i < U_i$, the actual benefit will be less than v_i. Also, there is no extra benefit in setting $u_i > U_i$ and this should be taken into account by the reconfiguration procedure.

As can be noted, reconfiguring the server parameters (Q_i, T_i) requires solving an optimization problem, where equation (1) is one of the constraints. The objective function of the problem will be described in the next section, since, as will be seen, it is associated to the version of the optimization problem being considered.

[1] Schedulability restrictions may be modified to support other scheduling policies. For example, this limit would be $n(2^{1/n} - 1)$ in rate monotonic

4 Reconfiguration Problem Definitions

In this section, after formulating the Integer Programming and the Linear Programming versions of the reconfiguration problem, a simple example is given. This example will be used later for illustration purposes.

4.1 Integer Programming Formulation

It is assumed here that, for each server S_i, $\kappa(i)$ values for U_i exist. In other words, let $K_i = \{1, 2, \ldots, \kappa(i)\}$ and $u_{ik} = Q_{ik}/T_{ik}$, $k \in K_i$, define the kth configuration of S_i for each of which there is a benefit A_{ik}. More formally, the optimization problem that must be solved by the reconfiguration mechanism is given as follows:

$$PD: \; fd = \text{Max} \sum_{S_i \in \mathcal{S}} \sum_{k \in K_i} A_{ik} x_{ik} \tag{2a}$$

S. to :

$$\sum_{S_i \in \mathcal{S}} \sum_{k \in K_i} u_{ik} x_{ik} \leq 1 \tag{2b}$$

$$u_{ik} = \frac{Q_{ik}}{T_{ik}} \tag{2c}$$

$$\sum_{k \in K_i} x_{ik} = 1, \; S_i \in \mathcal{S} \tag{2d}$$

$$x_{ik} \in \{0, 1\}, \; S_i \in \mathcal{S}, k \in K_i \tag{2e}$$

The variable x_{ik}, defined by equation (2e), indicates which configuration $k \in K_i$ is chosen for the S_i server. Exactly one configuration must be selected, which is reflected in restriction (2d). Restriction (2b) guarantees the schedulability of $\mathcal{S}$ according to the EDF policy. With no loss of generality, we assume that $u_{ik} \geq u_{ik-1}$ for every $S_i \in \mathcal{S}$ and $k \geq 2$. To model the possibility of cancellation of a server S_i, it is sufficient to define $A_{i1} = 0$ and $u_{i1} = 0$. In order to allow a new server S_i in the system at runtime, it is enough to set $A_{ik} > 0$ and $u_{ik} > 0$, $k = 1, \ldots, \kappa(i)$, as long as the designer has previously defined the maximum number of servers. We also assume that $\sum_{S_i \in \mathcal{S}} u_{i1} < 1$, otherwise the servers would not be schedulable or there would not be potential to optimize the applications.

Parameter A_{ik} defines the benefit attained by allocating u_{ik} units of computational resources to S_i, according to equation:

$$A_{ik} = \frac{\min(u_{ik}, U_i)}{U_i} v_i \,, \tag{3}$$

The objective function (3) has the following interpretation. If the system is able to allocate at least U_i to S_i, a benefit value of $v_i \geq 0$ is attained. The value v_i is to be derived from the knowledge of the system designer and represents one of the parameters used by the reconfiguration mechanisms. The value of v_i is not assumed to be static. The system/user can, at any time, change such values.

4.2 Linear Programming Formulation

The LP formulation is characterized by the absence of predefined configurations of the servers in $\mathcal{S}$. Thus, solving this version of the problem is to find values of u_i, $L_i \leq u_i \leq U_i$, such that there is a maximum benefit for the system, where L_i and U_i are lower and upper bounds on u_i, respectively. More formally, we define the following optimization problem:

$$PC : fc = \text{ Maximize } \sum_{S_i \in \mathcal{S}} A_i u_i \tag{4a}$$

$$\text{S. to :}$$

$$\sum_{S_i \in \mathcal{S}} u_i \leq 1 \tag{4b}$$

$$L_i \leq u_i \leq U_i, S_i \in \mathcal{S} \tag{4c}$$

Note that the objective function now means that the benefit A_i is accounted for per allocated server bandwidth unity. Similarly to the IP formulation, the system schedulability is ensured by equation (4b). Also there is no benefit in assigning $u_i > U_i$, which is ensured by equation (4c).

4.3 Illustrative Example

An illustrative example with $n = 4$ servers is presented in Table 1 assuming $\kappa(i) = 5$ predefined configurations for each server $S_i \in \mathcal{S}$, as required by the IP formulation. The benefit of each configuration is shown in Table 2.

As for the LP formulation, the illustrative example takes the form given in Table 3. The minimum and maximum bandwidth values for each server $S_i \in \mathcal{S}$, L_i and U_i, are taken from the values Q_i/T_i shown in Table 1. As can be noticed, the benefit values are now given per sever (or application) and represent the relative importance of a server in the system. The value of A_i shown in Table 3 was based on the values in Table 2 but normalized per unity of processor utilization.

Table 1. Predefined server configurations.

	Server S_i			
	$i=1$	$i=2$	$i=3$	$i=4$
u_{i1}	10/100	30/200	10/200	7/100
u_{i2}	15/50	25/100	08/40	9/60
u_{i3}	50/100	16/40	21/60	10/50
u_{i4}	35/50	110/200	30/50	15/50
u_{i5}	56/70	30/40	65/100	40/100

Table 2. Benefit per predefined configuration.

	Server S_i			
	$i=1$	$i=2$	$i=3$	$i=4$
A_{i1}	0.125	0.200	0.076	0.175
A_{i2}	0.375	0.333	0.307	0.375
A_{i3}	0.625	0.533	0.538	0.500
A_{i4}	0.875	0.733	0.923	0.750
A_{i5}	1.000	1.000	1.000	1.000

5 Solutions to the Reconfiguration Problems

This section presents the derived solutions for the formulated problems. Algorithms for the IP and LP formulations are given in Sections 5.1 and 5.2, respectively. Mode-change operations, to be carried out after the optimization results are obtained, are discussed in Section 5.3.

5.1 Integer Programming Formulation

The classic knapsack problem is trivially reducible to the reconfiguration problem, hence PD is NP-Hard [22]. It is well known that this problem can be solved by dynamic programming techniques. However, its pseudo-polynomial execution time prevents its use at runtime, prompting the design of approximation algorithms.

An algorithm that returns a solution that approximates the optimum is said to an approximation algorithm [18]. These algorithms can be useful in the search for a good solution when computation time is restricted, specially when dealing with NP-Hard problems. What follows is a general-

Table 3. Benefit values and bounds on server bandwidths.

	Server S_i			
	$i=1$	$i=2$	$i=3$	$i=4$
L_i	0.10	0.15	0.05	0.07
U_i	0.80	0.75	0.65	0.40
A_i	1.25	1.33	1.54	2.50

ization of approximation algorithms for the knapsack problem applied to the IP formulation. Both approximation algorithms described follow a greedy strategy.

5.1.1 Density Greedy Algorithm

The density greedy algorithm (DGA) has two distinct steps. First, it allocates the minimum resources possible to each server to ensure the feasibility of the reconfiguration. This resource pre-allocation is made necessary by equation (2d), that requires that each server receives a share of the system's resources. Second, it distributes the remaining resources to the servers following a non-increasing order of additional relative benefit $\hat{A}_{ik} = (A_{ik} - A_{i1})/(u_{ik} - u_{i1})$ as a primary ordering key and non-increasing order of relative resource demand $\hat{u}_{ik} = (u_{ik} - u_{i1})$ as a secondary ordering key.

By pre-allocating u_{i1} resource units to each server $S_i \in \mathcal{S}$, problem PD can be reformulated to include the minimum benefit for each server in the objective function and their minimum utilization in the restrictions to create a formulation equivalent to (2a)–(2e).

Let $\Omega = \{(i,k) : S_i \in \mathcal{S},\ k \in K_i\}$. Then, for a subset of pairs $\omega \subseteq \Omega$, let $S(\omega) = \{S_i : (i,k) \in \omega\}$ be the set of servers present in ω.

DGA$(\mathcal{S}, \{A\}, \{u\})$

Order the pairs of $\Omega - \{(i,1) : S_i \in \mathcal{S}\}$ in the sequence $\langle (i_1,k_1), \ldots, (i_{|\Omega|-|\mathcal{S}|}, k_{|\Omega|-|\mathcal{S}|})\rangle$ so that $\hat{A}_{i_p k_p} > \hat{A}_{i_q k_q}$ or $\hat{A}_{i_p k_p} = \hat{A}_{i_q k_q}$ and $\hat{u}_{i_p k_p} \geq \hat{u}_{i_q k_q}$ for every $p < q$
$\omega \leftarrow \emptyset$
$b \leftarrow 1 - \sum_{S_i \in \mathcal{S}} u_{i1}$
$t \leftarrow 1$
while $t \leq (|\Omega| - |S|) \wedge |\omega| < |\mathcal{S}|$ **do**
  if $S_{i_t} \notin S(\omega)$ and $(u_{i_t k_t} - u_{i_t 1}) \leq b$ **then**
    $\omega \leftarrow \omega \cup \{(i_t, k_t)\}$
    $b \leftarrow b - (u_{i_t k_t} - u_{i_t 1})$
  end if
  $t \leftarrow t + 1$
end while
for $S_i \in \mathcal{S} - S(\omega)$ **do**
  $\omega \leftarrow \omega \cup \{(i,1)\}$
end for
return ω

It is worth mentioning that the complexity of the algorithm DGA is $O(|\Omega| \lg |\Omega|)$, which is due to the need of ordering the elements of Ω whose size is at most nk^*, where $k^* = \max_{S_i \in \mathcal{S}} |K_i|$.

5.1.2 Modified Density Greedy Algorithm

The modified density greedy algorithm (M-DGA) returns the reconfiguration produced by DGA, ω, unless the reconfiguration of a server $S_{i'}$ in an execution mode k', (i',k'), induces an objective function with a larger value than that produced by the density greedy algorithm. In this case, M-DGA returns $\omega' = \{(i,1) : S_i \in \mathcal{S}, i \neq i'\} \cup \{(i',k')\}$.

M-DGA$(\mathcal{S}, \{A\}, \{u\})$

$(i',k') \leftarrow \arg\max_{(i,k)\in\Omega:k>1} \{A_{ik} - A_{i1} : u_{ik} - u_{i1} \leq 1 - \sum_{S_t \in \mathcal{S}} u_{t1}\}$
$\omega' \leftarrow \{(i,1) : S_i \in \mathcal{S}, i \neq i'\} \cup \{(i',k')\}$
$\omega \leftarrow DGA(\mathcal{S}, \{A\}, \{u\})$
if $fd(\omega') > fd(\omega)$ **then**
  return ω'
else
  return ω
end if

The quality of an approximation algorithm is usually stated as the performance ratio regarding the optimum solution, which is derived below.

Theorem 1. *Let I be an instance of the reconfiguration problem PD. The performance of the modified density greedy algorithm, denoted as M-$DGA(I)$, and the optimum performance, denoted as $OPT(I)$, are related by the following expression:*

$$M\text{-}DGA(I) \geq \frac{OPT(I) + \sum_{S_i \in \mathcal{S}} A_{i1}}{2} \tag{5}$$

Proof. Let $\omega^\star$ be the solution produced by M-DGA, where $\omega^\star = \omega$ if $fd(\omega) \geq fd(\omega')$ and $\omega^\star = \omega'$ otherwise. Let $(i_\omega, k_\omega) = \arg\max_{(i,k)\in\Omega:k>1} \{(A_{ik} - A_{i1})/(u_{ik} - u_{i1}) : (i,1) \in \omega\}$. Clearly,

$$OPT(I) \leq fd(\omega) + \frac{A_{i_\omega k_\omega} - A_{i_\omega 1}}{u_{i_\omega k_\omega} - u_{i_\omega 1}}(1 - \sum_{(i,k)\in\omega} u_{ik})$$

There are two possible cases. If $\omega^\star = \omega$, then:

$$OPT(I) \leq 2fd(\omega) - \sum_{S_i \in \mathcal{S}} A_{i1} = 2M\text{-}DGA(I) - \sum_{S_i \in \mathcal{S}} A_{i1}$$
$$\implies M\text{-}DGA(I) \geq \frac{OPT(I) + \sum_{S_i \in \mathcal{S}} A_{i1}}{2}$$

Otherwise, if $\omega^\star = \omega'$, then:

$$OPT(I) \leq 2fd(\omega') - \sum_{S_i \in \mathcal{S}} A_{i1} = 2M\text{-}DGA(I) - \sum_{S_i \in \mathcal{S}} A_{i1}$$
$$\implies M\text{-}DGA(I) \geq \frac{OPT(I) + \sum_{S_i \in \mathcal{S}} A_{i1}}{2}$$

Thus demonstrating the relation (5) between $OPT(I)$ and M-$DGA(I)$. □

Corollary 1. *The modified density greedy algorithm has relative performance* $R_{M\text{-}DGA} = 2$.

Proof. From the theorem above, we have that:

$$\frac{OPT(I)}{M\text{-}DGA(I)} \leq \frac{OPT(I)}{(OPT(I) + \sum_{S_i \in \mathcal{S}} A_{i1})/2} \leq 2 \quad (6)$$

Demonstrating that $R_{M\text{-}DGA} = 2$. □

We have extended a fully-polynomial approximation scheme (FPAS) for the standard knapsack problem to solve PD. This FPAS relies on a dual programming algorithm whereby the roles of objective and resource constraint are changed. This solution is not described here because $M\text{-}DGA$ produced nearly optimal solutions with little computational effort as compared with FPAS.

5.1.3 Illustrative Example

Table 4 shows the values of $\hat{A}_{ik}$, and the respective utilizations u_{ik}, obtained from the illustrative example presented in Tables 1 and 2. The execution of DGA gives the following result. The initial value of b is 0.63, set in line 3 of DGA. The first iteration of the `while` loop (lines 5-11) selects $\omega = \{(4,5)\}$, setting $b = 0.3$. Then, after the second iteration, $\omega = \{(4,5),(3,3)\}$ and $b = 0$, which means that no further improvement is possible. Then, after executing lines 12-15, the final value of $\omega = \{(4,5),(3,3),(1,1),(2,1)\}$, which gives a benefit value of $fd(\omega) = 1.863$. Carrying out M-DGA does not improve this result. Indeed, $(i',k') = (1,5)$, computed in line 1 of $M\text{-}DGA$, making $\omega' = \{(1,5),(2,1),(3,1),(4,1)\}$. However, $fd(\omega') = 1.451$. In fact, $fd(\omega) = 1.863$ turns out to be the optimum benefit value.

Table 4. Relative benefit per predefined configuration.

	$(\hat{A}_{ik}, u_{ik})$			
	$i=1$	$i=2$	$i=3$	$i=4$
$k=1$	(0.00, 0.10)	(0.00, 0.15)	(0.00, 0.05)	(0.00, 0.07)
$k=2$	(1.25, 0.30)	(1.33, 0.25)	(1.54, 0.20)	(2.50, 0.15)
$k=3$	(1.25, 0.50)	(1.33, 0.40)	(1.54, 0.35)	(2.50, 0.20)
$k=4$	(1.25, 0.70)	(1.33, 0.55)	(1.54, 0.60)	(2.50, 0.30)
$k=5$	(1.25, 0.80)	(1.33, 0.75)	(1.54, 0.65)	(2.50, 0.40)

5.2 Linear Programming Formulation

5.2.1 An Analytical Solution

Consider a set of n servers $S_i \in \mathcal{S}$ and let $L = \sum_{i=1}^{n} L_i$ and $U = \min(1, \sum_{i=1}^{n} U_i)$. Note that L and U represent the minimum and maximum bandwidth allocated to $\mathcal{S}$, respectively. The main idea behind the analytical solution to PC can be informally explained as follows. First, it is necessary to guarantee that each server has at least its bandwidth set to L_i. The remaining bandwidth, $U - L$, is then distributed to the servers in $\mathcal{S}$ in non-increasing order of benefit. Let $S_{(1)}, S_{(2)}, \ldots, S_{(n)}$ be an order on $\mathcal{S}$ so that $A_{(i)} \geq A_{(i+1)}$ for all $i < n$. For instance, if $\mathcal{S}$ is the set presented in Table 3, $S_{(1)} = S_4, S_{(2)} = S_3$ and so on. Thus, if $U_{(1)} - L_{(1)} \leq U - L$, then it is possible to allocate bandwidth $U_{(1)}$ to $S_{(1)}$. The allocation of the remaining bandwidth, i.e. $U - (L - L_{(1)} + U_{(1)})$, follows the same procedure regarding the second highest benefit server $S_{(2)}$ and so on. Let $S_{(i^*)}$ be the last server to which it was possible to allocate $U_{(i^*)}$ by this procedure. In this case, the bandwidth of server $S_{(i^*+1)}$ should be adjusted to the maximum allowed taking into consideration (a) the bandwidth $U_{(i)}$ allocated to $S_{(i)}$, $i = 1, \ldots, i^*$ and (b) that all the other servers $S_{(j)}$, $j = i^* + 2, \ldots, n$ stay with their minimum allowed bandwidth $L_{(j)}$. It is not difficult to see that the value of i^* can be computed as

$$i^* = \max_{i=0}^{n} \left\{ i : \sum_{j=1}^{i} (U_{(j)} - L_{(j)}) \leq U - \sum_{j=1}^{n} L_{(j)} \right\}$$

Notice that if $i^* = 0$, it is not possible to allocate $U_{(1)}$ to $S_{(1)}$. Also, if $i^* = n$, the bandwidth of all servers $S_{(i)} \in \mathcal{S}$ can be $U_{(i)}$. By using linear-programming theory it can be shown that the optimal solution to PC is:

$$u_{(i)} = \begin{cases} U_{(i)}, & \text{if } i \leq i^* \\ U - \sum_{j=1}^{i^*} U_{(j)} - \sum_{j=i^*+2}^{n} L_{(j)}, & \text{if } i = i^* + 1 \\ L_{(i)}, & \text{if } i \geq i^* + 2 \end{cases} \quad (7)$$

Notice that $\sum_{i=1}^{n} u_i = U$, which means that allocating bandwidth according to equation (7) maximizes the system bandwidth, a necessary condition for a solution $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ to be optimal.

Lemma 1. *Let* $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ *be a feasible solution to problem PC, defined by equations* (4a)-(4c). *If* $\sum_{i=1}^{n} u_i < U$, *then there is another feasible solution* $\mathbf{u}' = (u'_1, u'_2, \ldots, u'_n)$ *so that* $fc(\mathbf{u}') > fc(\mathbf{u})$ *and* $\sum_{i=1}^{n} u'_i = U$.

Proof. Let $\sum_{i=1}^{n} u_i = U' < U$. Hence, there must exist u_i and $\epsilon > 0$ so that $U' + \epsilon \leq U$ and $u_i + \epsilon \leq U_i$. Let $u'_j = u_j$ for all $j \neq i$ and $u'_i = u_i + \epsilon$. Since the objective function is monotonically increasing on u_i, it is clear that $fc(\mathbf{u}') > fc(\mathbf{u})$. Repeating this leads to $\sum_{i=1}^{n} u'_i = U$. □

The theorem below concisely shows the optimality of the derived solution.

Theorem 2. *Problem PC, defined by equations* (4a)-(4c), *has its optimum solution given by equation* (7).

Proof. The proof will be by contradiction. Consider a system $\mathcal{S}$ composed of n servers whose benefits are given in non-increasing order by $A_{(1)}, \ldots, A_{(n)}$. Assume that $\mathbf{u} = (u_1, \ldots, u_n)$ is the solution derived by equation (7) and it is not optimal. Thus, there must exist $\mathbf{u}' = (u'_1, \ldots, u'_n)$ so that $fc(\mathbf{u}') > fc(\mathbf{u})$. If $\alpha_i = u'_i - u_i$, we can write

$$fc(\mathbf{u}') - fc(\mathbf{u}) = \sum_{u'_{(j)} > u_{(j)}} \alpha_{(j)} A_{(j)} + \sum_{u'_{(k)} < u_{(k)}} \alpha_{(k)} A_{(k)} > 0 \tag{8}$$

If $u'_{(j)} > u_{(j)}$, we can conclude that $j \geq i^* + 1$ because $u_{(j)} = U_{(j)}$ for all $j < i^* + 1$. Also, if $u'_{(k)} < u_{(k)}$, $k \leq i^* + 1$ since $u_{(k)} = L_{(k)}$ for all $k > i^* + 1$. From these observations and from the fact that $A_{(i+1)} \leq A_{(i)}$ for all $i < n$ inequality (8) implies that

$$A_{(i^*+1)} \sum_{u'_{(j)} > u_{(j)}} \alpha_{(j)} + A_{(i^*+1)} \sum_{u'_{(k)} < u_{(k)}} \alpha_{(k)} > 0$$

By equation (7) it is known that $\sum_{i=1}^{n} u_i = U$. Also, by Lemma 1, we can assume that $\sum_{i=1}^{n} u'_i = U$. Hence,

$$\alpha = \sum_{u'_{(j)} > u_{(j)}} \alpha_{(j)} = - \sum_{u'_{(k)} < u_{(k)}} \alpha_{(k)}$$

and so $\alpha(A_{(i^*+1)} - A_{(i^*+1)}) > 0$, a contradiction. □

Once the optimum values of u_i are determined, the next step is to compute the server parameters (Q_i, T_i). This can be done by fixing one of the parameters, say T_i, so that $Q_i = u_i T_i$. For example, if T_i was related to a desired frame-per-second for a movie scene, Q_i would represent the corresponding bandwidth that leads to the optimum overall system benefit.

There is an aspect related to the fairness of the LP solution worth mentioning. For example, suppose that $A_{(i^\star)} = A_{(i^\star+1)} = A_{(i^\star+2)}$ and $L_{i^\star} = L_{i^\star+1} = L_{i^\star+2}$ for a given system $\mathcal{S}$. In this case, the solution given by equation (7) would let $u_{(i^\star)} = U_{(i^\star)}$ but $L_{(i^\star+2)} = u_{(i^\star+2)} \leq u_{(i^\star+1)} < u_{(i^\star)}$. This may not be fair for some applications since the same benefit values should mean the same bandwidth assignments as long as the bounds $L_i \leq u_i \leq U_i$ for all $S_i \in \mathcal{S}$. Thus, when fairness is to be considered, after obtaining the solution $\mathbf{u} = (u_1, \ldots, u_n)$ from (7), one may need to redistribute some server budgets. This budget redistribution needs to be carried out only if there are distinct servers $S_{(i)}$ and $S_{(j)}$ whose benefit values are $A_{(i)} = A_{(j)}$, for $i \leq i^\star + 1$ and $j \geq i^\star + 1$.

It is worth observing that the analytical solution given by equation (7) has time complexity $O(n)$ provided that the benefit values are in non-increasing order. This case represents scenarios where benefit values are statically assigned. Otherwise, when benefit values are assumed to change at runtime, ordering the servers is necessary and the time complexity becomes $O(n \lg n)$. The algorithm is not shown here since it is a straightforward implementation of (7).

5.2.2 Illustrative Example

When solving the reconfiguration problem as for the illustrative example shown in Table 3, one finds that $1 - \sum_{j=1}^{n} L_{(j)} = 0.63$ and $\sum_{j=1}^{1}(U_{(j)} - L_{(j)}) = 0.33$ but $\sum_{j=1}^{2}(U_{(j)} - L_{(j)}) = 0.93$ and so $i^\star = 1$. Hence, by equation (7), $u_{(1)} = 0.4$, $u_{(2)} = 0.35$, and $u_{(i)} = L_{(i)}$ for $i > 2$. The optimal objective is 1.8635.

5.3 Mode Change

Consider a system $\mathcal{S}$ with n CBS executing with a configuration $\mathbf{u} = (u_1, \ldots, u_n)$. Also, let $\mathbf{u}' = (u'_1, \ldots, u'_n)$ be another configuration, provided by one of the reconfiguration algorithms previously described. Assume that $\mathbf{u}$ is available at time t. There must be an efficient strategy to move $\mathcal{S}$ from $\mathbf{u}$ to $\mathbf{u}'$. To do that, we chose a simple but effective way by performing server budget replenishments at t.

Let (Q'_i, T'_i) represent the parameters of S_i according to $\mathbf{u}'$ and $c_i(t)$ and $d_i(t)$ be its current budget and deadline, respectively. A change of operation mode is carried out by applying standard CBS rules at t. If $c_i(t) \geq (d_i(t) - t)u'_i$, then (a) the maximum budget $c_i(t)$ is recharged to its maximum capacity Q'_i according to configuration $\mathbf{u}'$; and (b) the deadline $d_i(t) = d_i(t) + T'_i$. Otherwise, nothing is done, meaning that the current jobs in the queue of S_i are served with its current budget $c_i(t)$. When this budget is depleted, it is recharged to Q'_i as usual and the server deadline is updated accordingly [2].

6 Performance Analysis

For the practical evaluation of the methods defined above, they have been implemented along with an instance generator for the problem. A dynamic programming (DP) algorithm was implemented to produce optimal results against which the approximate solutions are compared. The DP solver has pseudo-polynomial time complexity and is used exclusively for comparison purposes. All algorithms were implemented in C++ and executed on Linux on an Intel Core2Duo 2.20GHz CPU with 2GB of RAM. Below, we evaluate the performance of the optimization algorithms, the effectiveness of the reconfiguration framework, and its integration with feedback control schedulers.

6.1 Assessment of the Optimization Algorithms

Problems of varying cardinality and number of configurations were generated to assess the performance of the methods, namely $25 \leq |\mathcal{S}| \leq 250$ and $3 \leq \kappa(i) \leq 10$. Obviously, a system with 250 tasks is unlikely, and those cases are considered here only for testing purposes. Configurations were generated using a uniform distribution for both utilization and benefit values. We average the solution time of 5 problems of same size to minimize measuring errors.

Figure 1 shows the execution times of the three methods, DGA, $M\text{-}DGA$ and the analytical solver for the LP formulation. While the server number used in the LP problems is the same as in the IP formulation, the problem is of lower complexity because of the continuous nature of the decision variables. A comparison with the DP algorithm (not shown in the graph) showed that these methods have an execution time approximately twenty thousand times shorter.

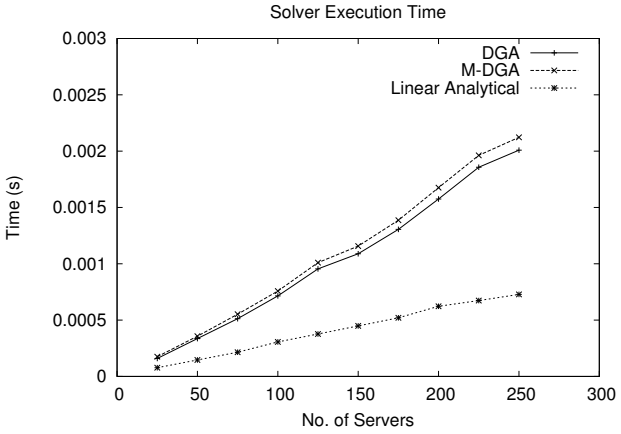

Figure 1. Solver execution time.

Figure 2 shows the quality of the solution returned by the approximation methods in absolute forms. The optimal solution showed in Figure 2 was obtained with dynamic programming, while the lower bound for the approximation and the upper bound of the optimal were calculated using inequality (5) of Theorem 1. It can be seen that the solutions returned by $M\text{-}DGA$ are much closer to the optimum than to the theoretic lower bound, induced by inequality (6).

6.2 Assessment of Reconfiguration Performance

To evaluate the performance of the reconfiguration infrastructure (reconfiguration algorithm and mode change policy) in a more realistic environment, the FFMPEG decoder was modified to generate a trace of release and computation times of a high definition video stream. A high variability in per-period utilization was observed, where utilization values were calculated by dividing each frame's

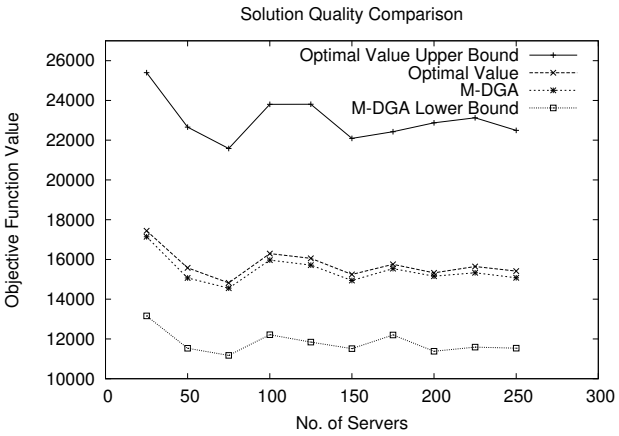

Figure 2. Absolute performance of the approximation algorithms.

computation time by the period between its release time and the release time of the next frame. While the mean overall utilization was of 19.7%, the single worst period showed a utilization of 88%. The mean computation time between all frames was 8.4ms, and the mean period is 42.5ms (a frame rate of approximately 23.5 frames per second).

In our simulation, three separate CBS receive jobs as specified by this trace. As a consequence, whenever the trace utilization raises above 33.3%, deadlines will be missed. This will allow the benefits of the mode change to be noticed directly by their effect on deadline misses. The period of each CBS was set to 42ms, the nominal period of the video stream, and their budgets were calculated through the utilization returned by the optimization solver.

We chose to present only the results obtained by the LP model. Similar but slightly worse results were achieved by using the IP formulation, which is expected due to the discrete nature of the IP formulation. Initially, all video decoding servers had equal benefit values $A_i = 1$ and therefore received equal processor shares of approximately 33%, in virtue of the fairness extension of the analytical solver, discussed in Section 5.2. The last 1% of CPU was attributed to a fourth server, dedicated to the solution of the optimization problem that arises from reconfiguration requests.

At 50 seconds, a reconfiguration was requested, raising the value of CBS number 3 so that $A_3 = 2$. This kind of reconfiguration request represents overload scenarios where an application timeliness is to be more critical than the others. The analytical solver in this particular case took 19.5us to complete, which corresponds only to 0.2% of the average time needed to decode a frame. After its completion, the new server parameters were applied according to the mode change policy described before, and the simulation ran until 100 seconds have elapsed. The same scenario was executed

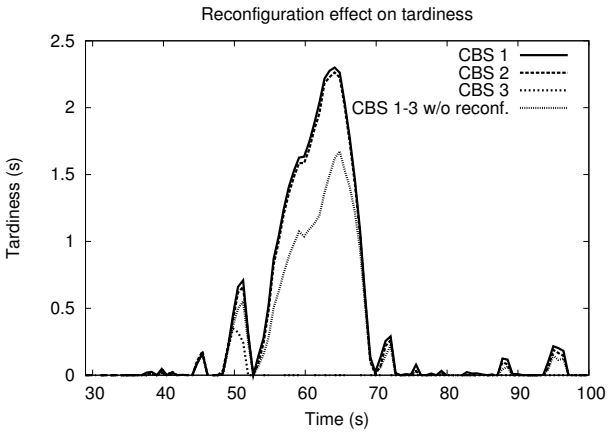

Figure 3. Tardiness observed during simulation and the effects of reconfiguration.

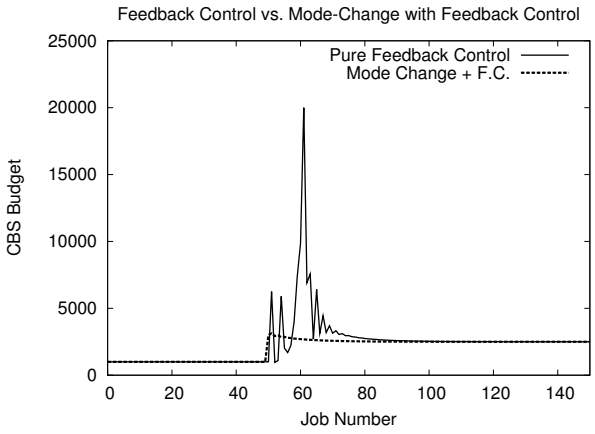

Figure 4. Comparison with Feedback Control

without reconfiguration for comparison.

Table 5 shows the CBS parameters and simulation results. Two QoS metrics were used: deadline miss ratio (D.M.R.) and mean tardiness (M.T.). Measurements were made before and after the reconfiguration request. L_i and U_i were set as approximately half and twice the mean utilization of the video decoding task, respectively. Mean tardiness, shown in seconds, was calculated as a measure of the severity of job delays. It can be seen that the deadline miss ratio of CBS 3 fell from 24.2% to 3.9% after the reconfiguration, and its mean tardiness was about 2% of that observed for the other servers. Indeed, the improvement obtained regarding CBS 3 is evident when comparing the results before and after the reconfiguration request.

Figure 3 shows the pattern of tardiness for those 3 servers, and also the pattern of tardiness for the same system without the reconfiguration. All servers are represented by the same line for the case without reconfiguration because they display nearly identical performance. It is worth noting that while CBS 1 and 2 had worse performance than on the simulation without mode change, the server deemed more important at the reconfiguration request ceased missing deadlines altogether 1.6 seconds after the new parameters were applied. This delay of 1.6 seconds is due to the backlog accumulated before the reconfiguration.

6.3 Integration with Feedback Control Systems

As mentioned in Section 2, an interesting approach to bandwidth reconfiguration is by means of feedback control. According to this approach, a control rule is used to dynamically adjust the server parameters, adapting the system to the current application demands based on the current scheduler performance. Scheduler performance is usually measured by the difference between what is required by the application (e.g. null tardiness) and what is delivered by the scheduler, which is called scheduling error. Adaptation can then be carried out in an autonomic manner.

In order to illustrate that feedback control scheduling is complementary to our reconfiguration infrastructure, we have integrated the mechanism proposed by Abeni et al. [3] with our approach. Figure 4 illustrates the effects of this integration in a scenario similar to that used by Abeni et al. Jobs 1 through 50 are scheduled without error with a CBS budget of 1000. Job 51 requires a budget of 2500, a step up in computation time. The feedback controller, when used by itself, oscillates and reaches the correct budget at job 142. To exemplify the usage of our infrastructure, the system designer overestimates the WCET and sets the budget for the new mode at 3000. The feedback controller then corrects this with much less oscillation and reaches the correct value of 2500 at job 120.

As can be seen from the figure, an adequate response to a drastic variation on utilization is achieved by the approach proposed in this paper, while feedback control may be used to fine tune the scheduler to account for small variations in task utilization. The best performance is obtained by using both approaches in conjunction. Furthermore, it is worth mentioning that if the benefit value differentiation can be determined by application designers, our approach allows for better handling of transient overloads which could otherwise cause all tasks to miss deadlines if only feedback control scheduling is implemented.

7 Conclusion

The dynamic reconfiguration of reservation-based scheduling parameters was addressed in this paper from the perspective of optimization problems. Integer Program-

Table 5. Simulation parameters and results.

			Before Reconfiguration				After Reconfiguration				Without Reconf.	
	L_i	U_i	A_i	u_i	D.M.R.	M.T.	A_i	u_i	D.M.R.	M.T.	D.M.R.	M.T.
CBS 1	0.065	0.395	1	0.328	0.246	0.293	1	0.295	0.562	0.516	0.422	0.202
CBS 2	0.065	0.395	1	0.328	0.239	0.296	1	0.295	0.711	0.542	0.416	0.198
CBS 3	0.065	0.395	1	0.328	0.242	0.296	2	0.395	0.039	0.011	0.418	0.201

ming (IP) and Linear Programming (LP) formulations were derived and solutions to both formulations were given and assessed by simulation. Two approximation algorithms for solving the IP formulation were presented and their solution quality was shown to be closed to that of an optimal solver. As for the LP formulation, an optimal solution was described. The running time of all derived solutions were found to be approximately twenty thousand times shorter than an exact solver based on dynamic programming.

Less restrictive task models, which include resource sharing and/or precedence relations among tasks, should be considered for future research. Also, it would be interesting to extend the reconfiguration problem to take into consideration restrictions among operation modes of different servers. The results presented here may well serve as a foundation for such developments in the field of scheduling for modern and adaptive real-time systems.

References

[1] L. Abeni and G. Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *Proc. of the 6th IEEE International Conference on Real-Time Computing Systems and Applications (RTCSA 99)*, pages 70–77. IEEE Computer Society, 1999.

[2] L. Abeni and G. Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27(2):123–167, 2004.

[3] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole. Analysis of a reservation-based feedback scheduler. In *Proc. of the 23rd IEEE Real-Time Systems Symposium (RTSS 02)*, pages 71–80. IEEE Computer Society, 2002.

[4] G. Beccari, S. Caselli, and F. Zanichelli. A technique for adaptive scheduling of soft real-time tasks. *Real-Time Systems*, 30(3):187–215, 2005.

[5] G. Buttazzo. Research trends in real-time computing for embedded systems. *ACM SIGBED Review*, 3(3), 2006.

[6] G. Buttazzo and L. Abeni. Adaptive workload management through elastic scheduling. *Real-Time Systems*, 23(1-2):7–24, 2002.

[7] M. Caccamo, G. Buttazzo, and L. Sha. Capacity sharing for overrun control. In *Proc. of the 21st IEEE Real-Time Systems Symposium (RTSS 00)*, pages 295–304, 2000.

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

[9] A. B. de Oliveira, E. Camponogara, and G. Lima. Dynamic reconfiguration of constant bandwidth servers. In *Proc. of the 10th Brazilian Workshop on Real-Time and Embedded Systems (WTR 08)*, pages 37–44. Brazilian Computer Society, 2008.

[10] J. Jehuda and A. Israeli. Automated meta-control for adaptable real-time software. *Real-Time Systems*, 14(2):107–134, 1998.

[11] 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.

[12] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek. On quality of service optimization with discrete QoS options. In *Proceedings of the IEEE Real-time Technology and Applications Symposium*, pages 276–286, 1999.

[13] G. Lima, E. Camponogara, and A. C. Sokolonski. Dynamic reconfiguration for adaptive multiversion real-time systems. In *Proc. of the 20th IEEE Euromicro Conference on Real-Time Systems (ECRTS 08)*, pages 115–124, 2008.

[14] G. Lipari and S. Baruah. Greedy reclamation of unused bandwidth in constant-bandwidth servers. In *Proc. of the 12th Euromicro Conference on Real-Time Systems*, pages 193–200, 2000.

[15] 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.

[16] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Marley. Performance specifications and metrics for adaptive real-time systems. In *IEEE Real-Time Systems Symposium*, pages 13–23, 2000.

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

[18] R. Motwani. Approximation algorithms. Book in preparation, 1992.

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

[20] C. Rusu, R. Melhem, and D. Mossé. Multi-version scheduling in rechargeable energy-aware real-time systems. *J. Embedded Comput.*, 1(2):271–283, 2005.

[21] M. Simoes, G. Lima, and E. Camponogara. A GA-based approach to dynamic reconfiguration of real-time systems. In *Proc. of the Workshop on Adaptive Embedded Systems (APRESS 08)*, 2008.

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

[23] P. Zhou, J. Xie, and X. Deng. Optimal feedback scheduling of model predictive controllers. *Journal of Control Theory and Applications*, 4(2):175–180, 2006.