Elastic Feedback Control

Marco Caccamo
Scuola Superiore S. Anna
Pisa (Italy)
caccamo@sssup.it

Giorgio Buttazzo
University of Pavia (Italy)
INFM Research Unit
giorgio@sssup.it

Lui Sha
University of Illinois
Urbana, IL 61801
lrs@cs.uiuc.edu

Abstract

In many real-time control applications, the task periods are typically fixed and worst-case execution times are used in schedulability analysis. With the advancement of robotics, flexible visual sensing using cameras has become a popular alternative to the use of embedded sensors. Unfortunately, the execution time of visual tracking varies greatly. In such environments, control tasks have a normally short computation time but also an occasional long computation time; therefore, the use of worst-case execution time is inefficient for controlling performance optimization. Nevertheless, to maintain the control stability, we still need to guarantee the task set even if the worst case arises. In this paper, we propose an integrated approach to control performance optimization and task scheduling for control applications where the execution time of each task can vary greatly. We create an innovative approach to elastic control that allows us to fully utilize the processor to optimize the control performance and yet guarantee the schedulability of all tasks under worst-case conditions.

1 Introduction

In many real-time control applications, the task periods are typically fixed and worst-case execution times are used in the schedulability analysis. This model is fine for many classical control applications where the execution time variations are small. However, with the advance of robotics, flexible visual sensing using cameras has become a popular alternative to the use of embedded sensors. Unfortunately, the execution time of visual tracking varies greatly.

For example, consider a ball and plate control system based on visual feedback. A ball on the plate has to follow a specified circle on the plate, which can be controlled by acting on roll and pitch rotations. To speed up the visual tracking process, predictive techniques are typically used to search for the ball in a small mobile window centered in the estimated ball position rather than searching the whole image. Normally, the ball is found in the small window and its position can be computed quickly. However, in most control applications there are occasional disturbances. If the disturbance makes the ball move outside of the predicted window, searching has to be extended in a larger area. This process may continue until, eventually, the entire plate is scanned.

In this example, the visual tracking task's computation time can be modeled as having a constant normal execution time, due to the searching operation in the small window. In addition, it has a bounded probabilistic execution time during tracking exceptions. Similar situations can be found in radar tracking where a search window is centered on the predicted location of the target.

It is worth noting that the search time is a part of the control loop, since the position information is needed in control computations. With a normally short (control loop) computation time, but an occasional long computation time, the use of worst-case computation times is inefficient. Nevertheless, to guarantee the control stability, we still need to close the control loop in time, even if the worst case arises.

If we reserve for the worst-case time, then most of the time there is a large amount of reserved but unused execution time budget. Although such an unused reserved time can be reclaimed for soft real-time aperiodic applications [8, 9, 10], such an aperiodic application may or may not exist in a given application environment. An alternative solution is to fully utilize the reserved budget to optimize the control performance in spite of the variation in computation time.

In digital control, the system performance is a function of the sampling rate. For a given controller design method, faster sampling will permit, up to a limit, a better control performance. So the idea is to increase the control loop frequency when the loop computation

time is short and slow down the frequency when the worst-case situation arises. However, there is a lower bound on the frequency for each task, f_i^{min}, the minimum frequency for each task τ_i, below which the performance is unacceptable - the control becomes unstable. In this formulation, $1/f_i^{min}$ represents the hard deadline that each instance of τ_i has to honor.

The frequency adjustment is particularly easy when the common method of digitized analog design is used, since this method does not require the change of control gains. The scheduler design method in this paper is also applicable to other control design methods. In this case, it involves the design of control gain scheduling in conjunction with scheduler design. However, application of this approach requires more work on the analysis of system stability. Since the focus of this paper is on the scheduler design, we shall assume that the digitized analogy controller design method is used for the system.

We refer the idea of dynamically adjusting the rates to optimize the control performance as an "elastic task tuning". However, a simple-minded implementation of this idea would not work. Consider the following example:

Task	$WCET_i$ *(ms)*	c_i^n *(ms)*	f_i^{min} *(Hz)*
τ_1	25	20	9.9
τ_2	25	20	20

Table 1. Task set parameters.

Example 1. A simple system is composed of two tasks whose parameters are shown in Table 1, where $WCET_i$ is the worst-case execution time in msec, c_i^n is the normal execution time in msec, and f_i^{min} is the minimal frequency in Hz. Suppose that when both tasks execute normally, the optimal frequency assignment for τ_1 is assumed to be 9.9 Hz, the same as the minimal frequency. Suppose also that the optimal frequency f_2^{opt} assignment for task τ_2 is 40 Hz, twice that of the minimal frequency of 20 Hz, whereas $f_1^{opt} = f_1^{min}$. Under this arrangement, these two tasks use 20% and 80% of the CPU respectively. The idea is that, should the worst case arise, task τ_2 could slow down from 40 Hz to 20 Hz to avoid overload. Unfortunately, this idea would not work.

As shown in Figure 1, both tasks start at time $t = 0$. At time $t = 101$, an exception is raised by the system because task τ_1 is asking to execute more than c_i^n units of time. Unfortunately, at this point, there is nothing task τ_2 can do to assist task τ_1. As a result, τ_1 will miss its deadline at $t_{ov} = 101$.

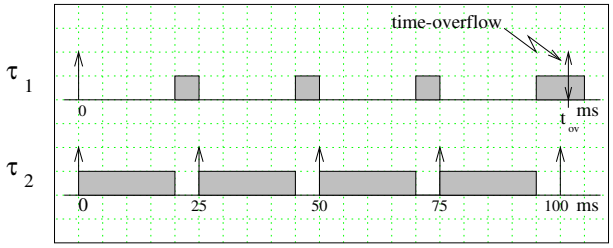

Figure 1. Example of time-overflow due to an exception.

To make the idea of elastic control work, the challenge is to develop a method that will adjust the task frequencies to produce the optimal system control performance, subject to the constraint of guaranteeing that when the worst case comes, no deadlines will be missed.

The problem of handling execution overruns in real-time systems has been recently addressed in the literature using various approaches. In [4], Gardner and Liu compared the behavior of three classes of scheduling algorithms for scheduling real-time systems in which jobs may overrun their allocated processor time, potentially causing the system to be overloaded. In their work, each task is characterized by a guaranteed execution time, which is zero for non real-time tasks, equal to the worst-case execution time for hard real-time tasks, and somewhere in between for soft real-time tasks. They introduced the Overrun-Server Method (OSM) and the Isolation-Server Method (ISM) as scheduling algorithms. Under OSM, a job is released for execution and scheduled in the same manner as it would be according to Deadline Monotonic (DM) [5] or Earliest Deadline First (EDF) [6] (in a fixed priority or dynamic priority environment, respectively). At the time of exception, the execution of the job is interrupted and the remaining execution time of the job is released as an aperiodic request to a server. Using the ISM technique, jobs are submitted as aperiodic requests to the server assigned to their task at the time of their release and execute completely under server control. However, these scheduling strategies can guarantee those jobs which do not generate exceptions meet their deadlines but are unable to guarantee a maximum response time for those which do.

Stankovic, Lu, Son and Tao, in [11, 12], describe another approach to increase the performance of a scheduling algorithm in unpredictable dynamic systems whose workloads cannot be accurately modeled. Usually, real-time systems are designed based on worst-

case workload parameters. When accurate system workload models are not available, such an approach can result in a highly underutilized system based on extremely pessimistic estimation of workload. In this work, they propose a new scheduling paradigm, called feedback control real-time scheduling, which defines error terms for schedules, monitors the amount of error, and continuously adjusts the schedules to maintain satisfactory performance. Using control theory methodology, they defined the percentage of tasks that miss their deadlines as the controlled variable and the requested CPU utilization as the manipulated variable. By doing this, the deadline miss ratio can be controlled by varying the admission strategy of tasks on-line. However, this technique rejects jobs to keep the system not fully loaded. The technique also is unable to isolate an overloaded task from the other tasks, affecting the performance of all the other tasks.

The problem of selecting the set of control task frequencies to optimize the system control performance subject to schedulability constraints was addressed by Seto, Lehoczky, Sha and Shin [7]. In this formulation, each control task τ_i is characterized by a *performance loss index* (PLI[1]) as a function of the sampling frequency. In fact, let J and $J_D(f)$ be the performance indices generated by a continuous-time control and its digital implementation at sampling frequency f, respectively; a performance loss index can be provided as $\Delta J(f) = |J_D(f) - J|$ which is convex and monotonically decreasing. Hence, if we note that $\Delta J(f)$ exponentially decreases with the frequency, then, for each control task, it can be approximated by $\Delta J_i(f_i) = \alpha_i e^{-\beta_i f_i}$, where f_i is the frequency of τ_i, α_i is a magnitude coefficient and β_i is the decay rate.

The performance loss index of the overall system $\Delta J(f_1, ..., f_n)$ is defined as $\Delta J(f_1, ..., f_n) = \sum_i w_i \Delta J_i(f_i)$, where w_i is a design parameter determined from the application. For instance, it can be the importance of the associated control system to have a better performance than the others.

Given the minimum frequency f_i^{min} and the worst-case execution time ($WCET_i$) of each task τ_i, Seto, Lehoczky, Sha and Shin provide an algorithm to compute the frequencies f_i^{opt} which minimizes the PLI of the system while guaranteeing the schedulability constraints (ensuring each task can meet its deadlines).

However, in [7], f_i^{opt} were computed based on $WCET$s. If the normal computation times c_i^n are much less than $WCET_i$, then f_i^{opt} can be too low.

In the following of this paper, we assume that each task has a given performance index function.

The rest of the paper is organized as follows. Section 2 introduces some terminology and assumptions we will use in the rest of the paper. Section 3 presents the main results consisting in a local approach for handling overruns, and an example is shown to illustrate the improvement in the performance we can achieve using our approach. Section 4 briefly recalls the Constant Bandwidth Server (CBS) algorithm and introduces an extension of it (CBS^{hd}) for efficiently schedule periodic tasks which generate overruns. Finally, Section 5 presents our conclusions and future work.

2 Terminology and assumptions

In this section we introduce the elastic control model developed for improving the performance of a real-time control system with large variations in computation time. We will assume that the task set is scheduled by the EDF algorithm, which assigns priorities inversely proportional to absolute deadlines.

For each task we introduce a normal computation time $c_i^n \leq WCET_i$ and we assume that it can be computed by analyzing the probabilistic distribution of the task computation time. Hence, each task is described by $\tau_i(c_i^n, WCET_i, f_i^{min}, \Delta J_i(f_i))$, where $WCET_i$ is the worst case execution time, f_i^{min} is the minimum frequency τ_i can execute at, and $\Delta J_i(f_i)$ is the PLI of τ_i. We require that each task must complete at or before its hard deadline $D_i^{hd} = 1/f_i^{min}$). Notice, however, that task τ_i will be normally scheduled using a dynamic deadline D_i less than or equal to D_i^{hd}.

The goal of this approach is to compute the optimal frequency f_i^{opt}, using the normal computation time c_i^n rather than $WCET_i$, while guaranteeing that each task will never miss its hard deadline D_i^{hd}. Furthermore, to simplify runtime management, we require that the worst case should be handled locally by each task, without affecting the frequencies of the other tasks.

We shall develop an algorithm which will give each task a computation budget as a percentage of the CPU. This budget will guarantee that every task will always meet its deadline. In addition, given the budgets, task frequencies can be adjusted in such a way that the control performance is optimal when tasks are using their normal execution times. The computation budgets can be implemented as a server that maintains the assigned budgets for each task.

In this model, each task can change its frequency dynamically depending on the current load. If $\tilde{d}_{i,j}$ is the soft deadline used by the server to schedule the job $J_{i,j}$ and its overrun (whenever it occurs), the next job $J_{i,j+1}$ will start at time:

[1] In the original formulation, the performance loss index was simply called performance index or PI. In the following, it will be called PLI for more clarity.

$$a_{i,j+1} = max(a_{i,j} + \frac{1}{f_i^{opt}}, \tilde{d}_{i,j}). \quad (1)$$

Hence, each task instance (job $J_{i,j}$) has a variable period $T_{i,j} = a_{i,j+1} - a_{i,j}$. Using this formalism and assuming that each instance has to complete by its fictitious deadline $\tilde{d}_{i,j}$, we need to guarantee that

$$\forall\, i,j \quad T_{i,j} \leq \frac{1}{f_i^{min}}. \quad (2)$$

In the next section we will show how to handle overruns correctly and how to perform an off-line guarantee of the task set. Notice that no restrictions are assumed on the number of overruns that a task can generate.

3 Local overrun handling

In this section, we introduce a simple policy for handling overruns. The proposed approach allows each task to handle its overruns locally, without affecting the frequencies of the other tasks. In this way, every time an overrun occurs, we only require a very small overhead to handle it.

In the following we assume that the optimal frequency f_i^{opt} has already been computed for each task τ_i, based on c_i^n, using the algorithm proposed in [7]. The method is based on reserving a bandwidth $U_i = f_i^{opt} c_i^n$ for each task τ_i. In normal conditions (without overruns), each task has a constant period $T_i = 1/f_i^{opt}$ and each task behaves as a classic periodic task[2]. When a generic job $J_{i,j}$ generates an overrun, a new deadline is computed for $J_{i,j}$ to avoid that the bandwidth consumed by τ_i exceeds its reserved bandwidth U_i.

Let $\tilde{d}_{i,j}$ be the soft deadline assigned by the server before the overrun occurs. That is,

$$\tilde{d}_{i,j} = a_{i,j} + \frac{1}{f_i^{opt}}.$$

If $J_{i,j}$ tries to execute more than c_i^n, the new deadline $\tilde{d'}_{i,j}$ of $J_{i,j}$ becomes:

$$\tilde{d'}_{i,j} = \tilde{d}_{i,j} + \frac{WCET_i - c_i^n}{U_i}. \quad (3)$$

The following example illustrates how an overrun is handled by the scheduling algorithm. The task set consists of two periodic tasks, τ_1 and τ_2, with minimum frequencies 1/20 and 1/12, worst case execution times 5 and 6, normal execution times 4 and 2, respectively. Moreover, let us suppose that the optimal frequencies

[2]Periodic tasks consist of an infinite sequence of identical activities, called instances or jobs, that are regularly activated at a constant rate.

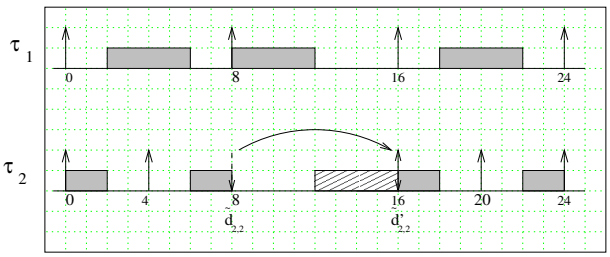

Figure 2. Example of overrun handled locally.

computed by the Seto, Lehoczky, Sha and Shin (SLSS) algorithm are $f_1^{opt} = 1/8$ and $f_2^{opt} = 1/4$. Therefore, each server has assigned a bandwidth $U_1 = U_2 = 0.5$.

Figure 2 shows what happens when τ_2 generates an overrun at time $t = 8$. Initially, the server assigns a soft deadline $\tilde{d}_{2,2} = 8$ to job $J_{2,2}$. At time $t = 8$, an overrun occurs and the server computes the new soft deadline $\tilde{d'}_{2,2} = \tilde{d}_{2,2} + (WCET_2 - c_2^n)/U_2 = 16$. After handling the overrun, the next job $J_{2,3}$ of task τ_2 arrives at time $a_{2,3} = 16$ and it will be executed again at its optimal frequency.

A simple necessary and sufficient condition states how it is possible to handle overruns locally; this result is expressed by the following theorem:

Theorem 1 *Given a set Γ of periodic tasks $\tau_i(c_i^n, WCET_i, f_i^{min})$ where each task is handled by a dedicated server with bandwidth U_i, then each task instance (job $J_{i,j}$) has a period $T_{i,j} \leq 1/f_i^{min}$ if and only if:*

$$\forall\, \tau_i \quad U_i \geq f_i^{min} WCET_i. \quad (4)$$

Proof.

If. Suppose equation (4) holds; then the worst case occurs when a task τ_k is scheduled with a bandwidth $U_k = f_k^{min} WCET_k$ and τ_k raises an overrun at the end of its period needing a computation time equal to its $WCET_k$. Supposing that a_k is the arrival time of the current instance of τ_k, an overrun equal to $WCET_k - c_k^n$ is detected at $t_{ovr} = a_k + c_k^n/U_k$. Now, we compute the response time R_k of τ_k including the interval of time required by the server to schedule the overrun:

$$R_k = \frac{c_k^n}{U_k} + \frac{WCET_k - c_k^n}{U_k} = \frac{c_k^n}{U_k} + \frac{WCET_k}{U_k} - \frac{c_k^n}{U_k} =$$

$$= \frac{WCET_k}{U_k} = \frac{WCET_k}{f_k^{min} WCET_k} = \frac{1}{f_k^{min}}.$$

Thus, the "*if* condition" follows.

Only if. By contradiction, suppose that a task $\tilde{\tau}$ exists such that equation (4) is not verified, that is: $\exists\ \tilde{\tau}\ \mid\ \tilde{U} < f^{min}WCET$ but still the task set is schedulable. If $\tilde{\tau}$ requests a computation time equal to its WCET, then the actual frequency $\tilde{f}$ of $\tilde{\tau}$ becomes: $\tilde{f} = \frac{\tilde{U}}{WCET} < f^{min}$ which means that the task set is not schedulable. A contradiction.

□

Using the result of Theorem 1, we can compute the optimal frequencies f_i^{opt} and guarantee a minimum frequency f_i^{min} for each task τ_i even in the presence of overruns. Like in the classical SLSS algorithm, in this approach a task set is guaranteed if and only if:

$$\sum_i f_i^{min} WCET_i \leq 1. \tag{5}$$

If the task set is feasible, a new lower-bound $\tilde{f}_i^{min}$ of frequency can be computed for each task τ_i in order to take overruns into account. In fact the parameter $\tilde{f}_i^{min}$ is defined as follows:

$$\forall\ \tau_i \quad \tilde{f}_i^{min} = \frac{f_i^{min} WCET_i}{c_i^n}. \tag{6}$$

Finally, the SLSS algorithm will be used to compute the optimal frequencies using c_i^n as computation time (instead of $WCET_i$) and $\tilde{f}_i^{min}$ as minimum frequency (instead of f_i^{min}) for each task τ_i.

Note that the SLSS algorithm is optimal among the algorithms which handle overruns locally (that is, without affecting the other tasks performance) even though the value of $\tilde{f}_i^{min}$ is used as minimum frequency instead of f_i^{min}. This is easy to prove, because the value $\tilde{f}_i^{min}$ represents the minimum frequency permitted to each task τ_i in order to handle overruns locally. Hence, in this model, each optimal frequency f_i^{opt} must be greater than or equal to $\tilde{f}_i^{min}$.

The performance cost of local handling provides us with the opening to consider global handling of overruns; that is, when an overrun occurs, instantaneously each task could decrease its frequency in order to free bandwidth for handling the overrun. However, this different approach is out of the scope of this paper.

3.1 An example

We illustrate the effect of the proposed technique on a bubble control system, which is a simplified model designed to study diving control in submarines. The same system was described by Seto, Lehoczky, Sha and Shin in [7]. Here, we describe a modified version of it, to emphasize the advantages achievable using our technique. The bubble control system considered here consists of a tank filled with air and immersed in the water. Depth control of the diver is achieved by adjusting the piston connected to the air bubble. In this example, a camera monitors the diver as sensor for getting its position.

Now, suppose that two such systems with different physical dimensions are installed on an underwater vehicle to control the depth and orientation of the vehicle, and assume they are controlled by one on-board processor. Hence, each control task is characterized by two different functions; the first one reads the image memory filled by the camera frame grabber and determines the actual position of diver, and the second one computes the next value of the control variable defined as the piston velocity. The first function of each control task is characterized by a variable computation time which depends on the current position of each diver; hence, we can assume that each task is characterized by a worst case execution time ($WCET_i$) and by a normal computation time c_i^n. The task set parameters are shown in Table 2, where, for each bubble control system i, $WCET_i$ (ms) is the control task worst-case execution time in each sampling period, f_i^{min} (Hz) is the lower bound on sampling frequency, and w_i is the weight assigned to system i.

The following data are given for the control design and scheduling problem: $\Delta J_i = \alpha_i e^{-\beta_i f_i}$, $i = 1, 2$, where the frequencies f_i must be determined.

Task	α_i	β_i	$WCET_i$ *(ms)*	f_i^{min} *(Hz)*	w_i
b_1	1	0.4	25	10	2
b_2	1	0.1	25	20	1

Table 2. Task parameters for the bubble control system.

A simple computation shows that the total CPU utilization of the overall bubble system is 75% when the minimum task frequencies are assigned. Supposing the total CPU utilization available for the bubble systems is 100%, Table 3 shows, at different values of c^n, the optimal frequencies computed from the SLSS algorithm and the resulting performance loss index of the overall system. Note that system performance increases as the performance loss index decreases. Moreover, $c^n = kWCET$ means that all normal computation times are reduced by that fraction. So, for instance, $c^n = 0.9WCET$ means that $c_1^n = 0.9WCET_1$ and $c_2^n = 0.9WCET_2$.

The results reported in Table 3 demonstrate that the control system performance significantly improves as the normal computation time is decreased with respect

c^n	f_1^{opt} (Hz)	f_2^{opt} (Hz)	ΔJ
$WCET$	12.16	27.84	0.0772
$0.9WCET$	13.05	31.40	0.0541
$0.8WCET$	14.16	35.84	0.0347
$0.7WCET$	15.59	41.56	0.0196
$0.6WCET$	17.49	49.17	0.0091
$0.5WCET$	20.16	59.84	0.0031

Table 3. Optimal frequencies and corresponding ΔJ for different values of c^n.

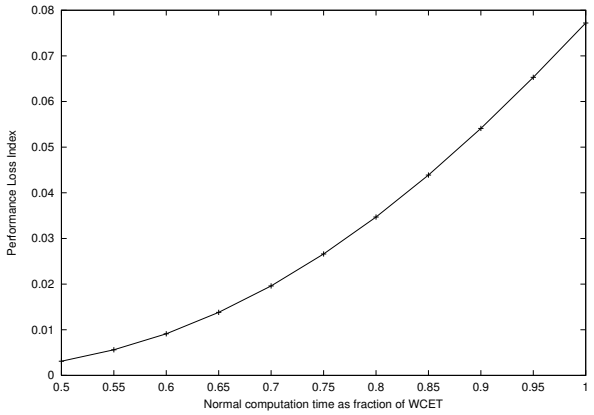

Figure 3. ΔJ vs. normal computation time c^n.

to the WCET, because tasks can run at higher frequencies. Figure 3 illustrates the relation between the performance loss index ΔJ and the value of c^n. In the graph, the normal computation time on the x-axis is expressed as a fraction of WCET. For instance, a value of 0.9 means that $c_1^n = 0.9WCET_1$ and $c_2^n = 0.9WCET_2$. Note that a little difference between c^n and WCET gives a significant gain in performance; for example, the performance loss index halves its value when each task has the normal computation time equal to 80% of its WCET.

In the following section we describe the characteristics of a service mechanism which allows us to efficiently implement the overrun handling algorithm described above.

4 Scheduling algorithm

In [1], Abeni and Buttazzo proposed a scheduling methodology, the *Constant Bandwidth Server* (CBS), devised for handling real-time tasks under a *temporal protection* mechanism. In order to provide task isolation, each task is assigned a fraction of the processor (a fixed *bandwidth*) and it is scheduled in such a way that it will never exceed its specified bandwidth, independently of its actual requests. This is achieved by assigning each task a suitable (dynamic) deadline, computed as a function of the reserved bandwidth and its actual requests. If a task needs to execute more than its expected computation time, its deadline is postponed so that its bandwidth is not exceeded. As a consequence, overruns occurring on task τ_a will only delay task τ_a, but will not steal the bandwidth assigned to the other tasks, which are then isolated and protected from reciprocal interference. In [2], the same authors present a statistical analysis for performing a probabilistic guarantee of soft tasks handled by the CBS algorithm.

In the following section we will briefly recall the CBS algorithm and its main properties.

4.1 The CBS algorithm

A CBS is characterized by an ordered pair (Q_s, T_s), where Q_s is the maximum budget and T_s is the period of the server. The ratio $U_s = Q_s/T_s$ is denoted as the server bandwidth.

At each instant, a fixed deadline $d_{s,k}$ and a budget c_s is associated with the server. Every time a new job $J_{i,j}$ has to be served, it is assigned a dynamic deadline $d_{i,j}$ equal to the current server deadline $d_{s,k}$. The current budget c_s represents the amount of computation time schedulable by the CBS using the current server deadline. Whenever a served job executes, the budget c_s is decreased by the same amount and, every time $c_s = 0$, the server budget is recharged to the maximum value Q_s and a new server deadline is generated as $d_{s,k+1} = d_{s,k} + T_s$.

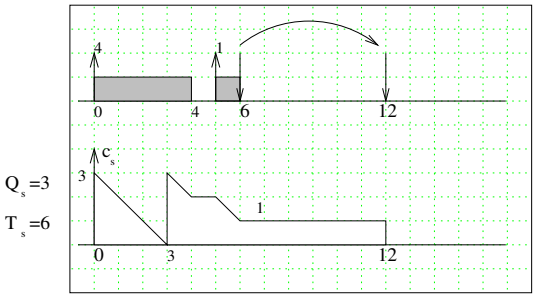

Figure 4. Example of a CBS server.

Figure 4 illustrates an example in which two jobs J_1 and J_2 are served by a CBS having a budget $Q_s = 3$ and a period $T_s = 6$. The first job arrives at time

$r_1 = 0$ and it is assigned a deadline $d_{s,1} = r_1 + T_s = 6$. Initially, c_s is equal to $Q_s = 3$; at time $t = 3$, the budget is exhausted, so a new deadline $d_{s,2} = d_{s,1} + T_s = 12$ is generated and c_s is replenished. At time $t = 4$, J_1 finishes and the server budget c_s is equal to 2 units; hence, the CBS is still available to schedule two units of computation time using the same server deadline $d_{s,2}$. At time $r_2 = 5$, the second job arrives and is served with the actual server deadline ($d_{s,2} = 12$).

In the model presented in this paper, each hard task τ_i must be scheduled with a reserved bandwidth U_i in order to isolate the effects of task overruns; however, we also need to guarantee that each task τ_i always has a frequency greater than or equal to f_i^{min}. In order to provide hard guarantee, for such tasks, we propose a new version of the CBS, called the CBS^{hd}.

4.2 The CBS^{hd} algorithm

The CBS^{hd} maintains the CBS properties and permits us to compute the dynamic deadline of each task in a more flexible way. In fact, each time an overrun occurs, the plain CBS recharges the budget to its maximum value and the task deadline is postponed by a fixed amount. In this way, however, the budget amount could be greater than the maximum overrun and the current task deadline would move too far away. In order to bound the task delay introduced by the overrun, the budget can be re-charged in a more fit way.

The CBS^{hd} rules are very similar to CBS rules. The new feature consists of a different way of re-charging the server budget and setting the new server deadline every time $c_s = 0$. In particular, the server budget is not always recharged to the maximum value Q_s. If $c^r_{i,j}$ is the remaining computation time of the current served job $J_{i,j}$ when the budget is exhausted, then every time $c_s = 0$, the following rule is applied for re-charge the budget:

if $(c^r_{i,j} \geq Q_s)$ {
  $c_s = Q_s$;
  $d_{s,k+1} = d_{s,k} + T_s$;
} **else** {
  $c_s = c^r_{i,j}$;
  $d_{s,k+1} = d_{s,k} + c^r_{i,j}/U_i$;
}

Using CBS^{hd}, the overrun is divided in chunks $H_{j,k}$, each characterized by a release time $a_{j,k}$ and a fixed deadline $d_{j,k}$. In this way, if the overrun is less than its maximum value, the current job will be scheduled more efficiently, avoiding computing the overrun deadline too pessimistically.

In the following we propose to schedule each task τ_i using a dedicated CBS^{hd} where the maximum budget Q_s is equal to c_i^n and the period of the server T_s is equal to $1/f_i^{opt}$.

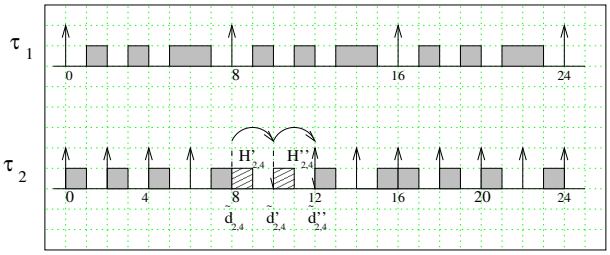

Figure 5. Example of overrun handled by CBS^{hd}.

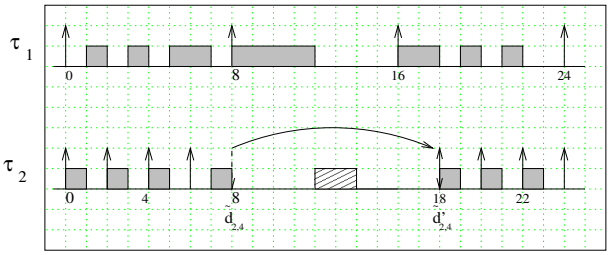

Figure 6. Example of overrun not handled by CBS^{hd}.

An example of chunks produced by a CBS^{hd} is shown in Figure 5. The task set of the example consists of two periodic tasks, τ_1 and τ_2, with minimum frequencies 1/20 and 1/12, WCETs 5 and 6, normal execution times 4 and 1, respectively. It is supposed that the optimal frequencies computed by the SLSS algorithm, are $f_1^{opt} = 1/8$ and $f_2^{opt} = 1/2$. Therefore, each CBS^{hd} is assigned a bandwidth $U_1 = U_2 = 0.5$, where the first server is characterized by the pair (4,8), and the second one by (1,2).

Figure 5 shows the schedule produced when τ_2 generates an overrun at $t = 8$. Initially, the CBS^{hd} assigns $J_{2,4}$ a soft deadline $\tilde{d}_{2,4} = 8$; at time $t = 8$, an overrun occurs and the server budget has to be recharged. Since $c^r_{2,4} = 5$ is greater than $Q_s = 1$, the server computes a new deadline $\tilde{d}'_{2,4} = \tilde{d}_{2,4} + T_s = 10$ for the first chunk $H'_{2,4}$. The overrun does not finish, so another chunk $H''_{2,4}$ needs to be scheduled in order to handle the whole overrun. However, at the end of the second chunk, the overrun completes and at time $t = 12$ the

next job $J_{2,5}$ arrives. Figure 6 shows how the same situation would be handled by a plain CBS. In this case, the overrun deadline is computed using equation (3). Note that, even if we insert a reclaiming mechanism, the period of $J_{2,4}$ cannot be less than 8 units of time, whereas the previous example shows that $T_{2,4} = 6$.

5 Conclusions

In this paper we presented a novel approach for increasing the efficiency of digital control systems in which the computation times of periodic activities have significant variations. The proposed method was proved to be particularly effective for those control activities, as visual tracking tasks, in which the worst-case computation time is much greater than the typical computation time required in normal operations.

The work presented in the paper integrates and extends two recent advances in real-time computing - the optimization of control performance subject to schedulability analysis and the Constant Bandwidth Server algorithm - to create an innovative approach to elastic control that allows us to fully utilize the processor to optimize the control performance and yet guarantee the schedulability of all tasks under worst case conditions.

It was shown that the proposed method is optimal among the algorithms which handle overruns locally (that is, without affecting the performance of the other tasks). An example was also described in order to illustrate the effect of the proposed technique, showing that the control system performance significantly improves as the normal computation time is decreased with respect to the WCET. In particular, the example described in Section 3.1 illustrates that a little difference between c^n and WCET provides a significant gain in performance: the performance loss index halves its value when each task has a normal computation time equal to 80% of its WCET. Finally, an improvement of the CBS algorithm was described in order to schedule the periodic control tasks more efficiently.

As a future work, we plan to investigate a technique for handling overruns globally, so that, when an overrun occurs, each task can decrease its frequency in order to create free bandwidth for handling the overrun. We are also investigating how the current methodology can be extended in the presence of resource constraints.

References

[1] L. Abeni and G. Buttazzo, "Integrating Multimedia Applications in Hard Real-Time Systems", *Proc. of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[2] L. Abeni and G. Buttazzo, "QoS Guarantee Using Probabilistic Deadlines" , *IEEE Proceedings of the 11th Euromicro Conference on Real-Time Systems*, York, UK, pp. 242-249, June 1999.

[3] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic Task Model for Adaptive Rate Control", *Proc. of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[4] M. K. Gardner and J. W.S. Liu, "Performance of algorithms for scheduling real-time systems with overrun and overload", *IEEE Proceedings of the 11th Euromicro Conference on Real-Time Systems*, York, UK, June 1999.

[5] J. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks", *Performance Evaluation*, 2:237-250, 1982.

[6] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard real-Time Environment", *Journal of the ACM* 20(1), 1973, pp. 40–61.

[7] D. Seto, J.P. Lehoczky, L. Sha and K.G. Shin "On Task Schedulability in Real-Time Control System", *Proc. of the IEEE Real-Time Systems Symposium*, December 1996.

[8] M. Spuri and G.C. Buttazzo, "Efficient Aperiodic Service under Earliest Deadline Scheduling", *Proc. of the IEEE Real-Time Systems Symposium*, San Juan, Portorico, December 1994.

[9] M. Spuri, G. Buttazzo and F. Sensini, "Robust Aperiodic Scheduling Under Dynamic Priority Systems", *Proc. of the 16th IEEE Real-Time Systems Symposium*, Pisa, Italy, December 1995.

[10] M. Spuri and G.C. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems", *The Journal of Real-Time Systems*, 10(2), 1996.

[11] J. A. Stankovic, C. Lu, S. Son and G. Tao, "The Case for Feedback Control Real-Time Scheduling", *IEEE Proceedings of the 11th Euromicro Conference on Real-Time Systems*, York, UK, June 1999.

[12] C. Lu, J. A. Stankovic, G. Tao and S. H. Son, "Design and Evaluation of a Feedback Control EDF Scheduling Algorithm", *Proceedings of the IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December 1999.