Real-Time control system analysis: an integrated approach

Luigi Palopoli Luca Abeni
Fabio Conticelli Marco Di Natale
Scuola Superiore S. Anna
Via Carducci 40 - I-56127 Pisa, Italy
E-mail: {palopoli,luca,contice,marco}@sssup.it

Giorgio Buttazzo
University of Pavia (Italy)
INFM - Pavia research unit
giorgio@sssup.it

Abstract

A typical approach for realizing digital controllers is to synthesize the control law in the continuous-time domain and then to implement it as a set of periodic threads complying with tight temporal constraints. The strict respect of all deadlines can often be obtained only by selecting low activation rates which determine a remarkable performance degradation. On the other hand, many control systems are known to tolerate a certain amount of deadline misses.

We realized a software tool which allows to numerically evaluate the quality of the control resulting from the scheduling. The tool has been applied to a robotic case study. Considering a meaningful set of trajectories, we have drawn experimental evidences that the use of soft real-time constraints on the threads leads to significant improvements in the system performance. The performance improvement is more evident if scheduling approaches like resource reservations schemes, able to separate the thread importance from its activation rate, are used.

1. Introduction

Many real-time digital controllers are realized in two steps. The first step consists of synthesizing a continuous-time control law tailored on the plant and designed to achieve system-level goals (stability, performance, robustness, etc.). The second step is dedicated to the design of a digital controller which implements the continuous time control law with a good approximation. The digital controller is intrinsically discrete-time, so the plant's output variables used to compute the controller output have to be periodically sampled. Digital-to-Analog converters, usually based on Zero Order Hold circuits (ZOH), transform the sequence of numbers generated by the digital controller into the continuous time signals used to drive the plant's actuators (see Figure 1). In this work, we consider computer-based embedded controllers operated by a multithreaded real-time operating system.

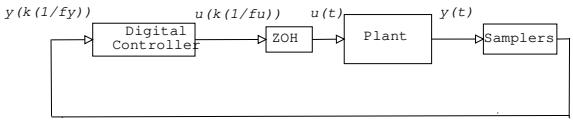

Figure 1. Block diagram of a digital controller.

Control theory usually assumes a highly deterministic timing behaviour. The most frequent assumption is that there is a negligible or fixed time interval between the acquisition of the sensor data and the release of the controller output is negligible or fixed. In reality constant delays like the ones deriving from ZOHs are, at least for linear systems, relatively easy to cater with in the controller design while stochastic response time variations deriving from data-dependencies and scheduling jitter are much more difficult to take into account. To match the control design requirements, the real-time community has endeavoured to provide a conception of time determinism on the side of the controller implementation based on the respect of deadlines associated to each thread execution (henceforth called *job*). This approach allows a clear separation between the design of the control law and its implementation simplifying the work of both the computer and the control engineers. However, there are some drawbacks, since the strict guarantee of every deadline, based on the worst case execution times of all threads can be often obtained only by selecting low activation rates. Besides, industrial practice reveals that the advantages of pushing the activation rates beyond the boundaries of hard real-time guarantees outweighs the price of occasional deadline misses, tolerated by many robust control schemes. As a matter of fact, releasing the hard deadline constraint can pay off an increase of nearly 50% in the performance index. On the other hand, we need design guidelines to evaluate alternative scheduling schemes (e.g. resource reservation) and fine tune the activation rates

and other scheduling parameters evaluating at each step the effect on a cost function. This implies to assess the performance of the scheduling algorithms in the control application domain by extensive simulation, while retaining a relaxed form of determinism on the thread schedule, such as the duration of the overload or a probabilistic guarantee.

The main contributions of this work are:

- an integrated approach for evaluating the performance of real-time control systems in the application domain against changes in the structure of the controller threads;
- a software tool that permits to perform integrated analysis on the performance of the control and on the parameters of the real-time scheduling;
- the evaluation of the performance of known scheduling approaches on a meaningful case study. In particular the work focuses on some important properties of resource reservation scheduling which proved to be very beneficial in the system design: the possibility of performing individual guarantees on each thread and the possibility of decoupling the importance of the thread from its activation rate.

The paper is organized as follows. Section 2 describes a general formulation of the problem along with an overview on the solution strategy. Section 3 presents a quick review of the real-time scheduling approaches with a particular focus on some of their relevant properties. Section 4 describes the software tool developed for this work. Section 5 shows the experimental results gathered on the case study. Finally, Section 6 states our conclusion and the future goals.

1.1. Related Work

Our approach is inspired by the integrated design for real-time control systems proposed in [23]. In this work an optimization procedure for the activation frequencies of control threads is proposed; the goal is maximizing the controller performance under schedulability constraints. The optimization is based upon the usage of cost function, as suggested in a previous work [27] which introduces a methodology to evaluate the hard deadlines of the threads. The influence of the computing delay on the controller performance is treated to a good extent in [26, 25]; in particular the authors consider the case of failures in updating the actuators (loss problem) and give an estimation of how many such events can be tolerated. These works are interesting because they show that control systems are able to tolerate a given number of data losses possibly caused by missed deadlines. Control techniques aimed at preserving the system's stability and performance against the effects of delays (in our case due to scheduling or network transmission delays) have been widely investigated for linear systems [4, 19, 20] , while some recent work has tackled the same problems in the nonlinear systems domain [18].

In the last years, hard real-time scheduling techniques (originally presented in [16]), have been extended to cope with different situations or constraints (i.e., arbitrary deadlines[15], generic fixed priorities [5], or resource constraints [24]). Different approaches based on relaxed constraints have been proposed in [22, 11] , but they have traditionally been limited to multimedia systems or signal processing. The advantage of applying flexible scheduling techniques in the domain of control applications is emerging quite clearly in recent real-time literature. In particular, [6] proposes an algorithm to dynamically change task periods in order to adapt them to the state of the system, while [17] use a feedback mechanism to adjust the system workload to the maximum that keeps the number of deadline misses below a desired threshold. Soft real-time techniques have also been considered in the design phase, for example see [7, 12].

As far as the use of software tools for assessing the performance of real-time controllers is concerned, an interesting work is presented in [10], where the authors propose a MATLAB toolbox to simulate a real-time scheduler in a SIMULINK block. The case study used in this paper has been shown with full details in [21] where we analyzed the performance loss when the multilevel control law is implemented by a set of hierarchical threads having a negligible duration. Finally, a noteworthy paper is [13]; the authors first map the classical control design parameters onto the end-to-end requirements of the controller and then apply the method of period calibration to derive the execution parameters of each thread so that the end-to-end requirements are respected.

2. Problem statement and solution strategy

We consider the basic problem of controlling a plant described by a set of differential equations:

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \\ \boldsymbol{y}(t) &= \boldsymbol{g}(\boldsymbol{x}(t),\ \boldsymbol{u}(t)) \end{aligned} \tag{1}$$

where $\boldsymbol{x}$ are the plant's state variables, $\boldsymbol{u}(t)$ are the input variables and $\boldsymbol{y}(t)$ are the output variables. Bold fonts are used to denote vectors. The aim is to synthesize a continuous-time and closed-loop control law $\boldsymbol{u} = \boldsymbol{C}(\boldsymbol{y}, t)$. Without loss of generality we assume that the control goal is to asymptotically regulate the controlled variables to $\mathbf{0}$. In order to realize such control law on a digital computer, it is necessary to specify the interface between the plant and the

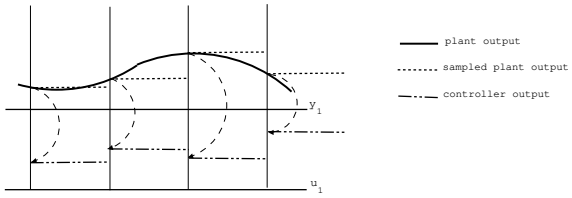

Figure 2. Ideal timing of a digital controller.

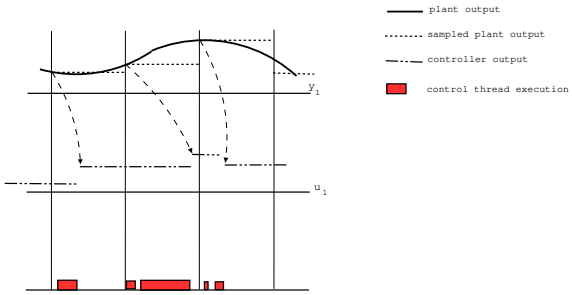

Figure 3. Timing of a digital controller in a multithreaded environment

computer and the implementation choices. The first problem is solved by using a set of samplers on the sensors and of ZOH on the actuators as shown in Figure 1, operated at fixed sampling rates. If f_i^y denotes the sampling frequency of the sensor on the i-th output variable, and f_j^u denotes the sampling frequency on the ZOH commanding the j-th actuator u_j,

$$u_j(t) = u_j(k\frac{1}{f_j^u}), \forall t \in [k\frac{1}{f_j^u},\ (k+1)\frac{1}{f_j^u}[.$$

In this framework, the digital controller produces the $u_j(k\frac{1}{f_j^u})$ outputs based on the sampled $y_i(k\frac{1}{f_i^y})$ and on the control law $\mathcal{C}$. Consider, as an example, the case of a single control loop. Assuming $f_1^y = f_1^u$, if the execution time of the program implementing the control law is neglected, the timing behaviour of the controller is depicted in Figure 2. At the beginning of each period, u_1 is emitted based on the last acquired value for y_1: the command is a piecewise constant updated with a fixed period, i.e. there is no scheduling jitter. Focusing on the implementation problem, a natural choice for the implementation of the controller is a set of periodic threads $\tau_1,\ \tau_2,\ \ldots,\ \tau_n$. Such threads have variable computation times, which can be modeled with a stochastic variable, and their jobs may experience delays due to scheduling. Going back to the example shown in Figure 2, a possible implementation for the controller consists of a periodic thread that reads the sensor data, computes the output and writes it to the actuator. Owing to the thread computation time and to the preemptions operated by higher priority threads, a variable delay can be introduced between the time a new sample for y_1 is read and the time the related value for u_1 is produced. A timing sequence which can actually occur using this scheme is shown in Figure 3. As a consequence, the time interval separating two outputs is not constant, causing an output jitter in the controller signal.

It is important observing that this implementation scheme introduces two types of delays. The first type is related to sampling; it is fixed and it does not depend on the threads execution time. The second depends on threads execution times and is affected by data dependent loops and the scheduling policy. In order to quantify the performance degradation, we introduce the following cost function [27]:

$$\mathcal{J} = \int_0^{+\infty} (\boldsymbol{y}^{\mathrm{T}} Q \boldsymbol{y} + \boldsymbol{u}^{\mathrm{T}} R \boldsymbol{u}) dt \qquad (2)$$

where Q and R are two positive definite matrices. The cost function is divided into two parts. The term $\boldsymbol{y}^{\mathrm{T}} Q \boldsymbol{y}$ accounts for the speed of $\boldsymbol{y}$ convergence to 0 which is the control goal. The shallower such convergence the greater the cost which is payed in the cost function. The term $\boldsymbol{u}^{\mathrm{T}} R \boldsymbol{u}$ accounts for the control energy which is spent to accomplish the goal. The more energy demanding a controller, the greater the price it pays in the cost function. Different importance can be attached to the output variable convergence or to the command energy by varying the ratio between the norms of Q and R.

In [23] the authors assume a set of independent control loops, realized by periodic threads, and hosted on a single CPU. A cost function similar to the one in Equation 2 is evaluated for each loop taking into account the sampling delays only. Consequently, the cost function turns out to be strictly decreasing with respect to the activation frequency and it is interpolated by a decreasing exponential (see Figure 4). An optimization procedure computes the optimal frequencies for a weighted sum of the cost functions considering the schedulability constraints and the threads worst case execution times.

A limitation of this approach is that many important control application are realized by interacting control loops. Moreover, when the threads execution times have a big variance, assuming the worst case may lead to a sensible underutilization of the CPU which in many cases outweighs the cost of occasional failures in updating the actuator commands. To cope with these shortcomings we propose to choose the scheduling rule and adjust the scheduling parameters according to the performance yielded in the control application domain. An essential role in this new design approach is played by the simulation tool described in Section 4.

When evaluating the scheduling performance, threads execution times are described by a stochastic behaviour. Hence, the cost function becomes a stochastic variable

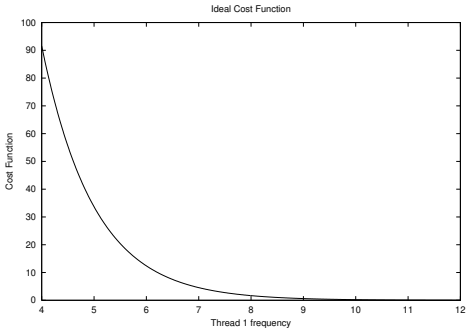

Figure 4. Performance index assuming null execution times.

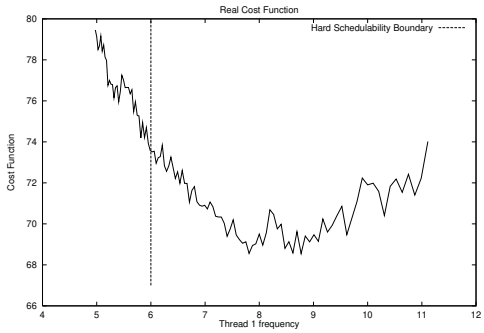

Figure 5. Cost function assuming real execution times.

which can be averaged through multiple runs, yielding:

$$\mathcal{J} = E\left[\int_0^{+\infty} (\boldsymbol{y}^{\mathrm{T}} Q \boldsymbol{y} + \boldsymbol{u}^{\mathrm{T}} R \boldsymbol{u}) dt\right], \qquad (3)$$

where $E[.]$ denotes the expectation. If the hard deadline hypothesis is released and the thread activation frequencies can be selected with greater flexibility, the cost function is no more necessarily decreasing. Figure 5 shows the cost function that we obtained in one of the experiments of Section 5. After traversing the boundary of hard schedulability, the function still improves since it benefits from the increased activation rate, paying a little price for the occasional deadline misses. If the activation rates are pushed too far, the overloads becomes permanent or semipermanent and the performance begins to degrade. This argument shows that, even if there is a clear advantage in releasing the hard deadline hypothesis, we still need to use some relaxed constraint, lest the predictability of the scheduling be completely dissipated. Therefore, in addition to the simulation tool, we propose the use of relaxed soft real-time constraints. In particular, in Section 3, we show that resource reservation scheduling approaches, providing further degrees of freedom in the scheduling choices, are very suitable for this kind of integrated design.

2.1. The case study

The proposed approach has been applied to a case study which can be considered as a representative of a wide class of nonlinear control applications. The controlled plant is a nonholonomic mobile robot, which is schematically illustrated in Figure 6. The goal of the controlled system is to track a moving target until an assigned displacement, expressed in a frame attached to the robot, is achieved. The control problem is analyzed with full detail in [21] and the proposed solution is based on the back-stepping approach, which leads to hierarchical nested subcontrollers. In the same paper, we showed a natural implementation scheme based on periodic threads communicating through asynchronous buffers. If the robot wheels are driven by motors

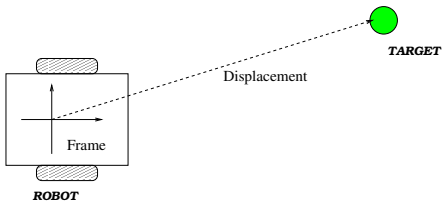

Figure 6. The case study

which can be commanded through the armature currents, the resulting controller comprises only two hierarchical levels. A simplified structure of the controller is reported in Figure 7. The data from the plant are acquired by two sensors. The first sensor is a camera used to determine the relative displacement between the robot and the target; it is sampled with period T_1. The second sensor is a pair of tachimetric encoders, used to read the wheels' angular velocity; this sensor is sampled with period T_2. The control law is implemented by two periodic threads: τ_1, τ_2. Thread τ_1 is activated with period T_1; it reads the relative displacement from a memory buffer and produces a reference value for the wheels' angular speed. Thread τ_2 is activated with period T_2; it reads the relative displacement and the wheels actual speed and produces a value for the currents to be applied to the DC motors of the wheels. The communication between the threads is assumed to be entirely asynchronous In order to keep an acceptable level of complexity for the analysis, the activity of extracting the relative displacement between target and camera is not performed by a separate thread, but it is accounted for in the computation time of thread τ_1. We remark that in this scheme the control loops are interacting; hence, it is not possible to apply the methodology proposed in [23].

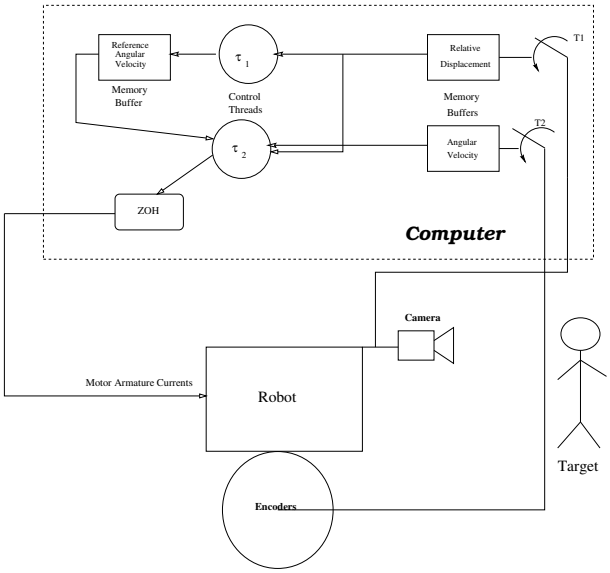

Figure 7. Control Scheme applied to the case study

3. Scheduling the Controller Threads

The controller is implemented as a set of concurrent periodic threads, and some form of determinism is required in the scheduling. Traditional real-time systems provide a deterministic schedule by guaranteeing that each job finishes within its period; in this section we will introduce a more relaxed form of guarantee, to improve the system performance.

For the sake of clarity, we give some basic definitions. A controller is implemented by a set $\Gamma = \{\tau_0, \ldots, \tau_n\}$, where each thread τ_i is a stream of instances, or jobs, $J_{i,j}$ with $j = 0, 1, 2, \ldots$. Each job $J_{i,j}$ arrives (is activated) at the beginning of a new period, at time $r_{i,j} = jT_i$ (where T_i is the thread period), executes for a time $c_{i,j}$, and finishes at time $f_{i,j}$. We denote by $C_i = \max_j\{c_{i,j}\}$ the Worst Case Execution Time (WCET) of thread τ_i and by $\overline{c_i}$ its mean execution time. Each job $J_{i,j}$ is assigned a deadline $d_{i,j}$ as $d_{i,j} = r_{i,j} + T_i$, and the goal of real-time scheduling algorithms is to guarantee that $J_{i,j}$ finishes within its deadline $d_{i,j}$ (that is, $f_{i,j} \leq d_{i,j}$). Such a guarantee is defined to be *hard* if it ensures that $\forall i, j,\ f_{i,j} \leq d_{i,j}$. If some jobs are allowed to miss their deadlines we talk about *soft* guarantee. In the sequel, we will briefly examine the type of guarantee provided by some well known scheduling approaches.

3.1. Hard Real-Time Scheduling

In [16] two classical algorithms are proposed: Earliest Deadline First (EDF), based on dynamic priorities, and Rate Monotonic (RM), based on static priorities. Using these algorithms, each job $J_{i,j}$ finishes within its deadline if the following admission test is verified:

$$\sum_{i=0}^{n} \frac{C_i}{T_i} \leq U_{lub} \tag{4}$$

with $U_{lub} = 1$ for EDF and $U_{lub} = n(2^{\frac{1}{n}} - 1)$ for RM. While for EDF this condition is necessary and sufficient, for RM it is only sufficient (if threads periods are harmonic, $U_{lub} = 1$ also for RM). Since the admission test is done on the worst case condition, the system can be underutilized if $C_i >> \overline{c_i}$. Consequently, in order to preserve the hard schedulability, very large periods have to be assigned to control threads, with the risk of achieving a poor performance.

While EDF and RM are known to be optimal when the system is not overloaded, they suffer in overload conditions. The RM algorithm in overload condition makes the longest period threads miss their deadlines, in favor of the shortest period ones. Therefore the only way for attaching importance to a thread is decreasing its period. Using EDF there is not a predictable way of attaching more importance to a thread with respect to the others in overload condition

3.2. Soft Real-Time Scheduling

An easy way to increase the system utilization can be to use the RM or EDF scheduling algorithms without the admission test in Equation 4. By making this choice, it is possible to select shorter periods for the control threads, at the price of no guarantee for the deadlines and possible overload situations. Clearly, the overload should be not permanent, otherwise the schedule becomes completely unpre-

dictable. Before getting on, we need to introduce a formal definition of *overload*: a scheduling system is said to be overloaded in the interval $(t_1,\ t_2)$ if the time demanded by all threads $\tau_i \in \Gamma$ in the interval is greater than $t_2 - t_1$.

If we define the time demanded by thread τ_i in interval $(0, t)$ as

$$D_i(0,t) = \sum_{j:d_{i,j} \leq t} c_{i,j}$$

the time demanded by all the threads in the system in $(0, t)$ is $D(0,t) = \sum_i D_i(0,t)$. Using this definition, the overload condition is said to be permanent if $\lim_{t\to\infty} \frac{D(0,t)}{t} > 1$. In this case, the schedule is not predictable and nothing can be said about any deadline.

A straightforward application of the queueing theory [14], shows that if the RM or EDF scheduling algorithms are used with a relaxed (optimistic) admission test changing Equation 4 into

$$\sum_{i=0}^{n} \frac{e_i}{T_i} \leq U_{lub}, \tag{5}$$

with $\overline{c_i} < e_i \leq C_i$, then the system shall not be subject to permanent overloads. Obviously, whenever the execution time of τ_i turns out to be lower than e_i, temporary overloads may occur.

3.3. Resource Reservations

In order to take full advantage from soft real-time scheduling, a method to control the number of deadlines missed by each thread is needed. This problem can be solved by scheduling techniques known as CPU reservation [1, 22]. Using a CPU reservation technique, each thread τ_i is assigned a pair (Q_i, T_i), where T_i is the thread period, and Q_i is the amount of CPU time reserved to τ_i in a period. If $\sum_j \frac{Q_j}{T_j} \leq U_{lub}$, τ_i is guaranteed to execute for Q_i time units every T_i, independently from the other threads.

As shown in [3], proportional share scheduling [28, 11] introduces similar concepts, but gives less control over CPU allocation.

In this work we focus on a particular reservation technique, called Constant Bandwidth Server (CBS)[1]. This approach is based on a global EDF scheduler that schedules threads according to their absolute *scheduling deadlines* assigned by a server mechanism (the Constant Bandwidth Server). Each thread is served by a dedicated CBS that enforces the reservation. The server mechanism assigns each job an initial deadline, which guarantees the reserved bandwidth B_i. A bandwidth isolation property is achieved by postponing the assigned deadline each time the thread demands more than the reserved bandwidth. Interested readers are referred to the cited paper for a precise and complete description of the algorithm and its properties.

With respect to the techniques shown in the previous sections, the use of the CBS algorithm (and of the resource reservation approaches in general) provides a wider control on the type of determinism that can be achieved. In [2] it is shown that if the Probability Distribution Function (PDF) of the execution times of τ_i is known and if $Q_i > \overline{c_i}$, then it is possible to compute the probability that job $J_{i,j}$ finishes within its deadline $r_{i,j} + kT_i$. Two things have to be noted: the first is that *each thread can be guaranteed independently from the others*: the second is that the guarantee can be based on different levels of criticality. In particular thread criticality can range from hard (the probability to respect the deadline $r_{i,j} + T_i$ is 1) to soft (the probability to respect a deadline depends on the reserved bandwidth). The guarantee is performed on each single thread τ_i: given $U_i(c) = P\{c_{i,j} = c\}$ (the execution times PDF), the probabilistic guarantee algorithms computes the probability $P\{f_{i,j} < r_{i,j} + \delta\}$ that a job finishes within a probabilistic deadline δ. This probability depends only on the PDF of the execution times of τ_i, on the bandwidth reserved to the thread and on the thread period, and does not depend on the other threads' parameters. Using this approach, if a thread set Γ is not schedulable with hard deadlines, it is possible to guarantee each thread $\tau_i \in \Gamma$ with a different criticality: in particular, if $Q_i \geq C_i$ (and $\sum_j \frac{Q_j}{T_j} \leq U_{lub}$) τ_i is guaranteed to respect all its deadlines. In this way it is possible to perform hard guarantee on some threads, and a relaxed guarantee on others.

Furthermore, if the PDF is limited (that is, $\exists \mu_i : c > \mu_i \Rightarrow U(c) = 0$), the task's completion times can be limited even when $Q_i < C_i$ (however, it is necessary that $Q_i > \overline{c_i}$). In other words, a thread τ_i served by a CBS has a minimum guaranteed execution rate.

Summarizing, the use of a CBS scheduling approach allows to raise the activation rates, while the effects of the transient overloads can be controlled by concentrating the deadline misses on the less important threads. In Section 5 we will show the effectiveness of this method and its simplicity.

4. The software tool

The integrated approach proposed in this paper requires the possibility of performing an integrated simulation of a plant and a computer based digital controller. In order to do this, we developed the Real-Time Controller Simulator (**RTCSIM**), a tool that uses a scheduling simulator and a differential equations integrator to simulate the actual implementation of the system.

The plant is described by a set of ordinary differential equations (ODE) depending on the physical parameters and on the external inputs. The digital controller is composed of the following elements: 1) a set of samplers, which period-

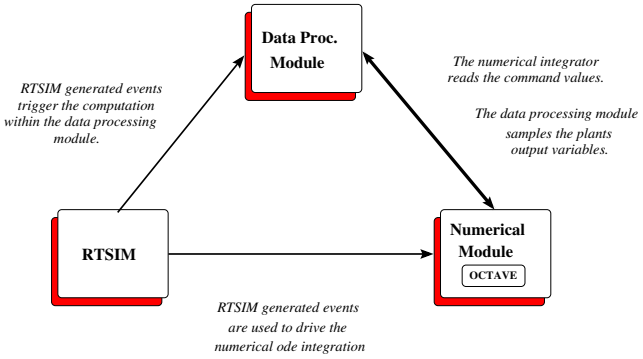

Figure 8. A software architectural view on RTCSIM

ically read the output of the plant; 2) a set of computing threads, described by statistical timing parameters which perform the computation and share the CPU according to a given scheduling policy; 3) a set of ZOHs, which convert the numerical outputs into piecewise constant signals to be used as input variables for the plant.

Our software tool is designed to simulate complex control systems (in the plant equation or in the control law structure). Since the statistical analysis requires multiple runs for each simulation, a fundamental requirement for the tool is high performance. The tool allows to test different scheduling policies and mutual exclusion protocols. An important requirement for our software is free availability; all its components are covered by the GPL license so that they can be freely downloaded from `http://hartik.sssup.it/~luigi/rtcsim` and used. The tool is composed of three independent components: a numerical module, a real-time systems simulator and a data processing module (see Fig. 8) .

The **numerical module** is used to *perform the ODE integration for the plant* and provides all the support for the linear algebra and other numerical computation. It is essentially based on the OCTAVE C++ [9] library. Different plants can be described by simply providing the implementation of a limited set of pure virtual methods; the most important of these methods is the differential equation which can be written in a very natural way using the mathematical abstractions provided by the numerical module.

To simulate the implementation of the controller in a multithreaded real-time system, we used a pre-existing library, called **RTSIM** [8]. RTSIM is a modular tool devised to *simulate distributed real-time systems*. We report some of its basic features referring the reader to the cited paper for further details. The structure of RTSIM is layered. The lowermost level, called METASIM, offers a set of basic classes for creating entities and events, along with a set of important utilities including execution tracers, statistics collectors and random variables generators. The upper levels of RTSIM offer a wide set of predefined entities for simulating real-time systems, such as pseudoinstructions, threads, kernels, servers etc. Real-time threads can be constructed as aggregations of pseudoinstructions having fixed or random duration; they can be handled by kernels endowed with different scheduling policies (Cyclic Executive, RM and EDF, Proportional share), or by aperiodic servers (Polling server, Sporadic Server , Constant Bandwidth Server, etc.). It is also possible to simulate mutually exclusive accesses to resources handled by different protocols (FIFO, Priority Inheritance/Priority Ceiling, Stack Resource Policy). Once a structure of threads has been specified along with their scheduling policy, RTSIM permits to construct a timed sequence of events associated with the activation and termination instants of the jobs, with the beginning and the end of a pseudoinstruction and so forth.

In the original RTSIM design, threads and their pseudoinstructions cannot perform "actual" computations; in other words no functional code can easily be associated with them, lest the simulator structure be radically changed. We found it convenient to introduce a third separate module (**the data processing module**) to fill in this gap: its goal is to *simulate the computational part of a digital controller*, RTSIM being focused on reconstructing its timing behaviour. The data module offers two types of components: computing units and storage units. In order to specify a computing unit, the programmer has to derive it from an abstract class and has to provide an implementation for three pure virtual methods. The first method acquires data from storage units and save them into the objects internal state. The second method is used to compute the output values based on the current state. The third one writes the data onto storage units. Three type of *storage units* exist: input buffers, memory buffers, output buffers. Input buffers are used to contain the data sampled from the plant; computing units are only allowed to read from them. Output buffers are models of a Zero Order Hold behaviour for the data applied to the plants simulated actuators; computing units are allowed to write into this type of storage units. Finally Read/Write buffers model memory location used by the computing units to communicate intermediate results one another; so they can be accessed by the computational units in both directions.

The three modules interact one another *via* well defined interfaces. The information flow between the numerical module and the data processing module is bidirectional. In fact, the numerical module needs to read the data held in the output buffer when integrating the differential equations. On the other hand, the data processing module needs to read the plant state, upon every sampling event, to update the content of the input buffers.

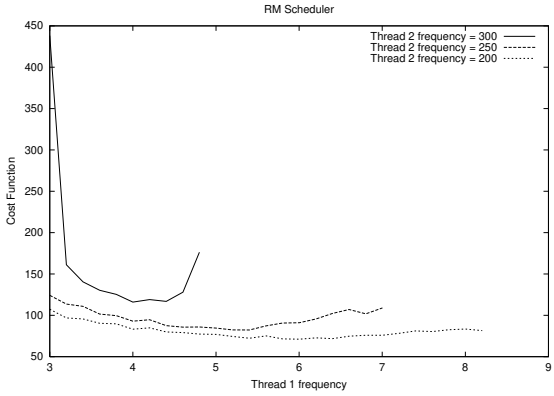

Figure 9. Cost function: RM scheduling.

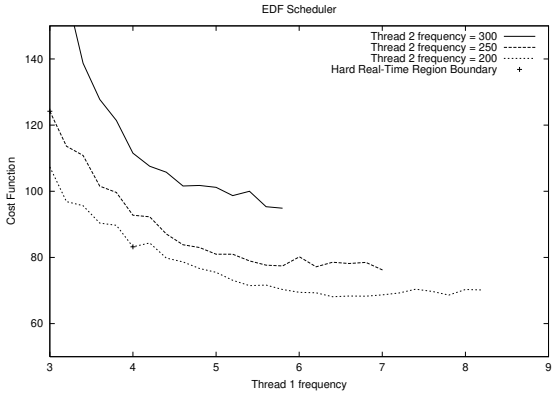

Figure 10. Cost function: EDF scheduling.

The relation between RTSIM and the data processing module can be explained considering that the computation units operations and the plant sampling have to be triggered at specific instants. Such instants are associated with the events produced by the RTSIM module. Finally the interaction between RTSIM and the numerical module is determined by the plant ODE integration: the RTSIM generated events are used to drive the integration.

5. Experimental Results

The simulation tool described in Section 4 has been applied to the case study described in Section 2.1 in order to show the influence of the scheduling choices on the system performance.

The robot physical parameters have been identified on a mobile robot provided by the IDEA s.r.l. company. We recall that the robot controller uses data having different timing characteristics. In particular the outer loop uses data coming from an external camera with a sampling period T_1, while the inner loop uses data sampled from a tachometer sensor with a period T_2. In the sequel we will also refer to the activation frequencies f_i defined as $\frac{1}{T_i}$. The loops are executed by periodic threads τ_1 and τ_2 according to the scheme shown in Figure 7. The threads' execution times have been assumed to be random variables uniformly distributed for the i-th thread in the range $[c_i, C_i]$, where c_i is the minimum execution time of the thread.

In order to produce a quantitative evaluation of the system's performance, we used the cost function in Equation 3. The system output is the difference between the measured relative displacement between the robot and its target and the desired one, both evaluated in the robot frame.

The simulations have been performed assuming the robot must track a still target having coordinates $\boldsymbol{x}_g = [3,\ 1.5]^{\mathrm{T}}$ in the inertial frame, while the initial robot coordinates are $\boldsymbol{x} = [0, 0]^{\mathrm{T}}$. The final desired displacement in the robot's frame is $\boldsymbol{x}^{(d)} = [0.2,\ 1]^{\mathrm{T}}$. Choosing the weight matrices $Q = I$, $R = 10^{-3}I$, the continuous time controller yields a value for the cost function equal to 35.1.

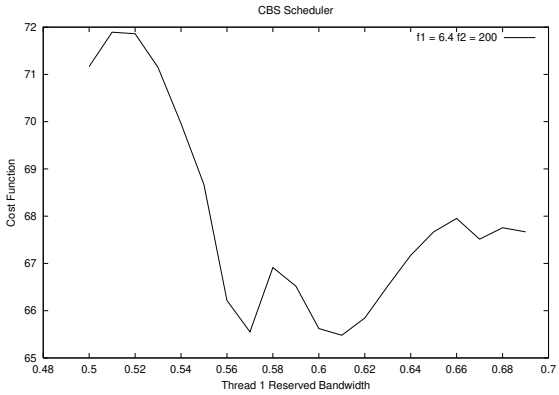

Figure 11. Cost function: CBS scheduling.

A preliminary study [21] (performed assuming null execution times for the controller threads) revealed that the choice of f_1 has an influence on the system's performance remarkably greater than the choice of f_2. An intuitive explanation could be that, although τ_2 always finishes within its deadline, it is forced to use outdated data if τ_1 is delayed. For this paper, assume the thread execution times as $c_1 = 40ms$, $C_1 = 100ms$, $c_2 = 1ms$, $C_2 = 2ms$. In order to model computation activities different from the control threads - for example data logging - we used a periodic thread having a fixed computation time $C_3 = 4ms$ and an activation period $T_3 = 20ms$. This activity "steals" a processor utilization factor $u_3 = \frac{C_3}{T3} = 0.2$.

In a first set of experiments, we tested the influence of the threads activation rates on $\mathcal{J}$ when the RM and EDF scheduling algorithms are used. All the experiments have been performed by running 30 repetitions of each simulation, and estimating the *mean* cost function (as expressed in Equation 3) along with the 95% confidence interval. The

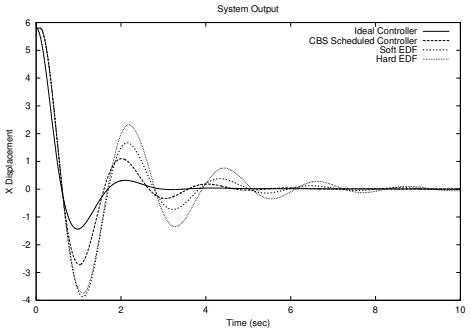

Figure 12. System output (x displacement)

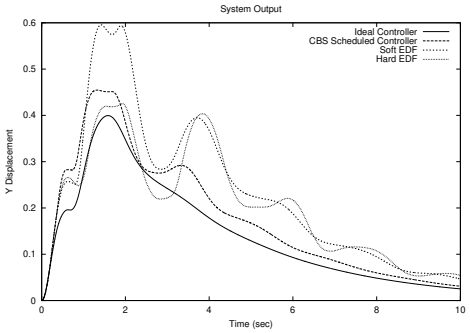

Figure 13. System output (y displacement)

ranges of possible frequencies f_1 and f_2 have been selected according to considerations on the physical characteristics of the device and were set to $[300Hz,\ 500Hz]$ for f_2, while the upper bound for f_1 was chosen to respect Equation 5.

Figure 9 shows the variation of the cost function with respect to f_1 and f_2 when the RM algorithm is used to schedule the controller threads: high values for f_2 ($f_2 = 300$) determine a significant performance loss, since the schedulability is jeopardized. Decreasing f_2 to $200Hz$, the system workload diminishes, and subsequently τ_1 is offered the possibility of respecting a greater number of deadlines; as a result, the system performance improves. This result is in accordance with our preliminary study and our considerations on the system. As a matter of fact, the direct influence that f_2 has on $\mathcal{J}$ is lower than its indirect influence caused by the deadline misses on the τ_1 thread.

A similar effect can be observed for τ_1: if its activation frequency is too high, the cost function rises (especially in combination with big values for f_2); on the other hand, τ_1 is more important for the controller, so decreasing its activation rate beyond a reasonable limit leads to a performance degradation. The optimal value for the cost function ($\mathcal{J}^{RM} = 71.16$) is achieved by choosing ($f_1 = 6, f_2 = 200$).

Figure 10 plots the cost function variations when the EDF algorithm is used. The behaviour is similar to the one observed using the RM algorithm, although the values obtained for the cost function are lower, meaning that a better performance can be achieved.

It is worth noting that the optimal frequency assignment under EDF results in $f_1 = \frac{1000ms}{156ms} = 6.4Hz$ and $f_2 = \frac{1000ms}{5ms} = 200Hz$, giving $\mathcal{J}^{EDF} = 68.126801$. Since $\frac{100}{156} + \frac{2}{5} + \frac{5}{20} = 1.291 > 1$ this configuration is outside the hard guarantee region. In Figure 10, the boundary of the hard guarantee region is shown: it can be seen that the best performance that can be obtained using real-time scheduling is $\mathcal{J} = 83.208$, with $f_1 = 4Hz$, and $f_2 = 200Hz$.

In all experiments, we obtained a confidence interval smaller than 3% of the mean value.

Using a reservation technique, such as the CBS, an additional degree of freedom - represented by the reserved amount of resource - can be used to further improve performance. As an example, we executed a simulation run by scheduling the controller threads with dedicated CBS servers, assigning frequencies $f_1 = 6.4Hz$, and $f_2 = 200Hz$ which resulted to be optimal using EDF. Since we assumed the presence of residual computation activities using 0.2 of the system bandwidth, the bandwidth B_1 and B_2, associated to τ_1 and τ_2 respectively, have been chosen according to the relation: $B_1 + B_2 = 0.8$. Thus, we varied B_1 from 0.5 to 0.7 and assigned $B_2 = 0.8 - B_1$. The resulting cost function is plotted in Figure 11: by increasing the bandwidth assigned to τ_1 (the most important task) the performance increases, until the limit $B_1 = 0.61$ is reached. Increasing B_1 beyond this limits slows down τ_2 too much and $\mathcal{J}$ degrades. The optimal cost function is $\mathcal{J}^{CBS} = 65.4793 < \mathcal{J}^{EDF}$: this fact confirms that the reservation approach is beneficial. We remark that using the CBS scheduler, it is possible to attach more importance to the τ_1 thread regardless of its activation rate. The plot is not exactly convex but it presents a peak for $B_1 = 0.58$: this effect seems rather to be a consequence of the strong nonlinearity of the system.

In order to visually assess the performance improvement obtained using soft real-time techniques (and reservations in particular), Figure 12 and 13 show as an example the dynamics of the x and y coordinates of the displacement obtained using different kind of scheduling. In particular, the figures show the dynamics obtained using an ideal controller (with 0 execution times for the controller threads), the CBS scheduled controller with $f_1 = 6.4Hz$, $f_2 = 200Hz$ and $B_1 = 0.61$, the soft EDF controller with $f_1 = 6.4Hz$ and $f_2 = 200Hz$ and the hard EDF controller with $f_1 = 6Hz$ and $f_2 = 200Hz$. The plots show a performance improvement on the CBS scheduled case more

evident than it would be expected from the numerical differences in the cost function.

6. Conclusions

In this paper we presented a novel approach for realizing software architectures targeted to real-time controllers. We realized a software tool for the joint simulation of the plant dynamics and of the scheduling of the controller threads. This tool enabled us to investigate the effects of the scheduling policy on the controller performance. In particular we showed, on a meaningful case-study, that the performance of classical real-time scheduling approaches (based on a strict respect of every deadline) can be poor, if compared to soft real-time approaches able to tolerate transient overload situations. The only evidences of this fact are, so far, experimental. As a future work we aim at achieving a deeper analytical insight into this problem.

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE Real Time Systems Symposium*, Madrid, Spain, December 1998.

[2] L. Abeni and G. Buttazzo. Qos guarantee using probabilistic dealines. In *Proceedings of the IEEE Euromicro Conference on Real-Time*, York, England, June 1998.

[3] L. Abeni, G. Lipari, and G. Buttazzo. Constant bandwidth vs proportional share resource allocation. In *Proceedings of the IEEE International Conference on Mutimedia Computing and Systems*, Florence, Italy, June 1999.

[4] K. Åstrom and B. Wittenmark. *Computer Controlled Systems, (3rd ed.)*. Prentice Hall, 1997.

[5] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, September 1993.

[6] G. Buttazzo, L. Abeni, and G. Lipari. Elastic task model for adaptive rate control. In *Proceedings of the IEEE Real Time Systems Symposium*, Madrid, Spain, December 1998.

[7] M. Caccamo, G. Buttazzo, and L. Sha. Elastic feedback control. In *Proceedings of the IEEE Euromicro Conference on Real-Time*, Stocolm, Sweden, June 2000.

[8] A. Casile, G. Buttazzo, G. Lamastra, and G. Lipari. Simulation and tracing of hybrid task sets on distributed systems. In *Real Time Computing Systems and Applications*, 1998.

[9] J. Eaton et al. *http://bevo.che.wisc.edu/octave*.

[10] J. Eker and A. Cervin. A matlab toolbox for real-time and control systems co-design. In *Proc. of The Real-Time Computiong Systems and Applications*, Hong Kong, China, December 1999.

[11] P. Goyal, X. Guo, and H. M. Vin. A hierarchical cpu scheduler for multimedia operating systems. In *2nd OSDI Symposium*, October 1996.

[12] D.-I. Kang, R. Gerber, and M. Sakena. Performance-based design of distributed real-time systems. In *IEEE Real-Time Technology and Applications Symposium*, pages 2–13, June 1997.

[13] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, and H. Shin. Visual asessment of a real-time system design: a case study on a cnc controller. In *Proceedings of the IEEE Real-time Systems Symposium*, 1996.

[14] L. Kleinrock. *Queuing Systems*. Wiley-Interscience, 1975.

[15] J. Leung and J. W. Whitehead. On the complexity of fixed priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4), 1982.

[16] C. L. Liu and J. Layland. Scheduling alghorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), 1973.

[17] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Design and evaluation of a feedback control EDF scheduling algorithm. In *Proceedings of the IEEE Real Time Systems Symposium*, Phoenix, Arizona, December 1999.

[18] S. Monaco and D. Normand-Cyrot. *Nonlinear systems*, volume 3, chapter 5. Chapman & Hall, 1996.

[19] J. Nilsson, B. Bernhardsson, and B. Wittenmark. Some topics in real-time control. In *Proc. American control conference*, Philadelphia, June 1998.

[20] J. Nilsson, B. Bernhardsson, and B. Wittenmark. Stochastic analysis and control of real-time systems with random time delays. *Automatica*, 34(1):57–64, 1998.

[21] L. Palopoli, F.Conticelli, and B. Allotta. Multi-level stabilizing control of nonholonomic vehicles and its multirate digital implementation. In *Proc. of IEEE Conference on Robotics and Automation 2000 (ICRA 2000)*, Stanford, USA, April 2000.

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

[23] D. Seto, J. Lehoczky, L. Sha, and K. Shin. On task schedulability in real-time control systems. In *IEEE Real Time System Symposium*, December 1996.

[24] L. Sha, R. Rajkumar, and john P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE transaction on computers*, 39(9), september 1990.

[25] K. Shin and X. Chui. Computing time delay and its effects on real-time control systems. *IEEE Transactions on Control Systems Technology*, 3(2):218–224, June 1995.

[26] K. Shin and H. Kim. Derivation and application of hard deadlines for real-time contol systems. *IEEE Transactions on Systesms, Man and Cybernetics*, 22(6):1403–1412, November/December 1992.

[27] K. Shin, C. Krishna, and Y. Lee. A unified method for evaluationg real-time computer controllers and its application. *IEEE Transactions on Automatic Control*, AC30(4):357–366, April 1985.

[28] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceedings of the IEEE Real Time Systems Symposium*, December 1996.