New On-Line Preemptive Scheduling Policies for Improving Real-Time Behavior

Mathieu Grenier
LORIA-INRIA
Campus Scientifique, BP 239 54506 Vandoeuvre-lès-Nancy - France
grenier@loria.fr tel: +33 3 83 58 17 68, fax: +33 3 83 58 17 01

Abstract

In real-time systems, schedulability is mandatory but other application-dependent performance criteria are most generally of interest. We first define the properties that a "good" real-time scheduling algorithm must possess. Then, we exhibit a class of easily-implementable policies that should be well suited to various applicative contexts because, in our experiments, these policies provide good trade-off between feasibility and the satisfaction of the application-dependent criteria. The study is illustrated in the framework of computer-controlled systems that are known to be sensitive to various delays induced by resource sharing. We evaluate criteria others than feasibility by simulation. An extended version of this article, comprising feasibility analyses for the considered policies, is accepted in the ETFA conference, see [10].

1. Introduction

Context of the paper. In real-time systems, feasibility of the task set is the basic requirement, but, usually, other criteria besides feasibility are of interest. A prominent example are computer-controlled systems [2] where it is well-known that other temporal characteristics than deadline respect affect the performances of the controlled system [16, 2].

Goal of our paper. The goal is here to take into account others criteria than feasibility in the scheduling design. The aim of the study is to find on-line scheduling policies that are well suited to the satisfaction of application dependent criteria. In the following, we will illustrate our approach through computer-controlled systems where, most usually, reducing delays and their variabilities improve the performances.

Related Work. Many studies have been dedicated to find scheduling solutions that improve the performance of computer-controlled systems (a more complete state of the art can be found in [10]).

To better fit to the processing requirements of a control system, new task models have been conceived. In [7, 5], it is proposed that control tasks are Subdivided in three different parts: sampling, computation, actuation. Sampling and actuation Sub-tasks are assigned a high priority in order to reduce the jitters.

Another solution, is to adjust the parameters of the tasks to achieve the desired goals. In [3], the worst-case end-of-execution jitter is minimized by choosing appropriate deadlines. In [8], initial offsets and priorities are adjusted to reduce jitter by minimizing preemption.

Improvements can also be brought by well choosing the parameters of the scheduling policies. In [9], a priority allocation scheme is proposed to reduce the average response time while, in [15], the problem of choosing scheduling policies and priorities on a Posix 1003.1b compliant operating system (OS) is tackled.

Finally, another way is to create new scheduling policies. In [1], the scheduler is synthetized as a timed automata from the Petri net modeling the system and the properties expected from the system. In [11], also starting from a Petri net model of the system, an optimal scheduling sequence is found by examining the marking graphs of the Petri net.

Our approach. In this paper, we propose a technique for building new on-line scheduling policies that ensure feasibility and perform well with regard to application dependent criteria such as the ones that are crucial in computer-controlled systems. We do not merely tune the parameters of a scheduling policy, as the priorities [9] for FPP scheduling, but tune the scheduling algorithm itself. The main advantage with regard to [1] and [11] is the lower complexity. Finally, the approach could be used in conjunction with task splitting schemes [7, 5]. Our proposal is made of two distinct steps:

1. define the characteristics that a "good" real-time scheduling policies must possess. This class of good policies constitutes the search space of our problem,
2. explore the search space for finding policies that perform well in terms of feasibility and with respect to

the other criteria.

Organization. In Section 2, the model of the system and the assumptions made are presented. In Section 3, the requirements for an acceptable policies are defined. Then, in Section 4, we define the criteria beside feasibility and the search space of the scheduling policies. In Section 5, the experimental results are presented.

2. System model

This study deals with the non-idling scheduling of periodic tasks on a single resource which can be either a CPU or a communication network.

Task model. The task model is very similar to the one used in [13]. A periodic task τ_i is characterized by a triple $(C_i, \overline{D}_i, T_i)$ where C_i is the worst case execution time (WCET), $\overline{D}_i$ the *relative deadline* (*i.e.* maximum allowable response time of an instance - equal for all instances of the same task) and T_i the *inter-arrival time* between two instances of τ_i. The release time of j^{th} instance of the i^{th} task is denoted by $A_{i,j}$ ($(A_{i,1})$ is thus the initial offset of the task).

Defining scheduling policies through priority functions. Priority functions, introduced in [14], is a convenient way of formally defining scheduling policies. The priority function $\Gamma_{k,n}(t)$ indicates the priority of an instance $\tau_{k,n}$ at time t. The resource is assigned, at each time, according to the *Highest Priority First* (HPF) paradigm.

Function $\Gamma_{k,n}(t)$ takes its value from a totally ordered set $\mathcal{P}$ which is chosen in [14] to be the set of multi-dimensional $\mathbb{R}$-valued vectors $\mathcal{P} = \{(p_1, ..., p_n) \in \mathbb{R}^n \mid n \in \mathbb{N}\}$ provided with a lexicographical order, with convention $(p_1, ..., p_n) \prec (p'_1, ..., p'_{n'})$ iff ($\exists i \leq \min(n, n') \vdash p_i = p'_j, \forall j < i$ and $p_i > p'_i$) or ($n' < n$ and $p_j = p'_j, \forall j \in [1, ..., n']$). Example: $(3, 4, 5) \leq (3, 2, 5)$, $(2, 4, 5) \geq (3, 2, 5)$, $(2, 3) \geq (2, 3, 1)$.

Most real-time scheduling policies can be defined easily using priority functions. For instance, the priority of an instance $\tau_{k,n}$ under preemptive Earliest Deadline First (EDF), is $\Gamma^{EDF}_{k,n}(t) = (A_{k,n} + \overline{D}_k, k, n)$ (the last two coordinates are needed to ensure decidability, see definition 3).

Besides providing non-ambiguous definition of the scheduling policy, priority functions enable us to distinguish classes of scheduling policies and to derive generic results that are valid for whatever the policy belonging to a certain class. The next Section presents requirements that a scheduling policy must fulfill.

3. "Good" scheduling policies

An arbitrary priority function does not necessarily define an implementable scheduling of interest for real-time computing. In this Section, we precise the requirements expected from an acceptable policy (termed "good" policy in the following). A "good" policy must meet a certain number of criteria, which are needed for the policy to be implemented in a real-time context.

Decidable policies. Policies are needed to be *decidable*: at any time t, there is exactly one instance of maximal priority among the set of active instances (*i.e.* instances with pending work). This concept of decidability was introduced in [14]. For instance, the last two components of $\Gamma^{EDF}_{k,n}(t) = (A_{k,n} + \overline{D}_k, k, n)$ ensure decidability.

Implementable policies. For being *implementable* in practice, a policy must induce a finite number of context switches over a finite time interval. This first condition, called "piecewise order preserving", was exhibited in [14]. Furthermore, components of the priority vectors have to be representable by machine numbers. In the following, coordinates of a priority vector belong to the set of rational numbers $\mathbb{Q}$.

"Shift temporal invariant" policies. In this study, for the sake of predictability of the system, we are only interested in scheduling policies such that the relative priority between two instances does not depend on the numerical value of the clock: relative priority must remain the same if we "shift" the arrival of all instances to the left or the right. The policy is thus independent of the value of the system's clock at start up time. We call such policies *shift temporal invariant* (STI) policies. EDF is a STI policy since the priority between two instances only depends on the offset between arrival dates and on relative deadlines.

We have defined a minimum set of requirements that a "good" policy must fulfill in the context of real-time computing.

4. Performance criteria and study domain

In this section, the performance criteria take into account in this study are precised. Then, among the set of all good policies, we define the particular class of scheduling policies considered in this study.

4.1. Performance criteria

We consider periodic control loops where the control algorithm is modeled by a periodic task τ_k with period T_k (i.e. the sampling period). In classical control theory, the main parts of a control loop are sampling, control computation and actuation. Some assumptions are made: the data collection from sensors (i.e. sampling) is assumed to be done at the beginning of each instance (at time $B_{k,n}$), the computation of the control law is performed in a constant time C_k, and actuation, that is the transmission of output data to the actuators, is done at the end of execution of each task instance (at time $E_{k,n}$).

Specific delays of control loops have been identified to be of particular importance for the stability of the system, and, more generally, for its performances (see, for instance, studies in [16, 2, 6]). These delays are:

- *Input-output latency* (or *response time*) of an instance $\tau_{k,n}$ is the time elapsed between the sampling and the actuation (equal to $E_{k,n} - B_{k,n}$ with our notations),
- *Sampling interval* is the time interval between two consecutive sampling instants (*i.e.* $B_{k,n+1} - B_{k,n}$),
- *Sampling latency* of an instance $\tau_{k,n}$ is the time elapsed between the theoretical sampling time (*i.e.* $A_{k,n}$), and its actual occurrence (*i.e.* $B_{k,n}$).

In classical discrete control theory, input-output latencies and sampling intervals are assumed to be constant with no sampling latency. In practice, when resources are not dedicated to a single control loop, these delays exists and greatly impact the performances (see [16, 12]). The aim is thus to keep these delays and their variabilities (jitters) as close as possible to the assumptions made by the theory.

4.2. Search space

In order to be able to analyze the scheduling policies and to define more precisely the search space, this one is limited to the class of "Arrival Time Dependent" policies.

"Arrival Time Dependent" policies. In the following, our domain of study is a Sub-class of Time Independent policies (i.e. $\forall t\ \Gamma_{k,n}(t) =$ constant) that we call *Arrival Time Dependent Priority* (ATDP).

Definition 1 *An Arrival Time Dependent policy is a policy whose priority function can be put under the form*

$$\Gamma_{k,n}(t) = (A_{k,n} + p_k, k, n) \tag{1}$$

where p_k*:* $k \mapsto \mathbb{Q}$*.*

p_k is an arbitrary function, which returns a constant value for all instances of task τ_k. For instance, for EDF, the value returned by p_k is equal to the relative deadline $\overline{D}_k$.

In the following, to achieve the desired goal (feasibility + performance), experiments will be done within a Sub-class of ATDP policies having a priority vector of the form:

$$\Gamma_{k,n} \quad = \quad (A_{k,n} + c.C_k + d.\overline{D_k}, k, n) \tag{2}$$

where $d \in [0, 1]$ and $c \in [0, 100]$ (i.e. $p_k = c.C_k + d.\overline{D_k}$ in definition 1). A point $\mathcal{C}$ in our search space is a policy defined by a priority function of the form of equation 2.

Motivations for Arrival Time Dependent policies. First of all, ATD policies are "good" scheduling policies:

- decidability is ensured by the last two components of the priority vectors,
- the policies are implementable in the sense of definition 3; the priority functions are "piecewise order preserving" due to constant priority over time and they can be represented by machine numbers.

Secondly, this sub-class of ATD policies has been chosen because we expect that it contains policies providing a good trade-off between feasibility and the satisfaction of the other criteria important for control systems (see 4.1). EDF actually belongs to this class and policies whose priority function is "close" to EDF are expected to have nearly the same behavior in terms of schedulability. On the other hand, introducing a term dependent of the execution time should help to improve the other criteria. Indeed, it has been shown that Shortest Remaining Processing Time First is optimal for average response times in various contexts (see [4]).

5. Experiments

Experiments in this study are performed in the framework of computer-controlled systems. Corresponding performance criteria were presented in §4.1 and the space of scheduling policies is defined in §4.2.

Case study. In this experiments, several control tasks sharing a CPU where the initial offsets of the tasks are not known. The task sets are generated with a global load randomly chosen in the interval $[0.8, 0.9]$ with $\overline{D}_i = T_i$.

We consider the case where the policy is tuned for a particular application (see [10] for case where the policy has to be efficient on average). The aim is here to find the policy that leads to a feasible schedule and that provides the greatest improvement for the other criteria. The search space, defined by equation 2, is exhaustively search with steps of granularity $d = 0.1$ and $c = 0.5$ (the search space comprises approximately 2000 policies).

Evaluation of criteria. Feasibility of our policies is computed with the generic feasibility analysis presented in [10], which is an extension of the EDF response time analysis.

The two performance criteria are the improvement in percent over EDF of the average sampling latency and of the average sampling interval jitter (measured as the standard deviation of the sampling intervals). The values of the criteria are computed with the data collected during simulation runs, as to our best knowledge there is no analytic technique. A given criterion is evaluated for a policy as the average value of the criterion for all tasks (for each task set 20 different initial offsets are simulated).

Results. Results are depicted on figure 1. Each point is the average improvement over 100 runs (only the best policy at each run is considered), where a run is defined by a task set randomly generated with an average load of 0.85.

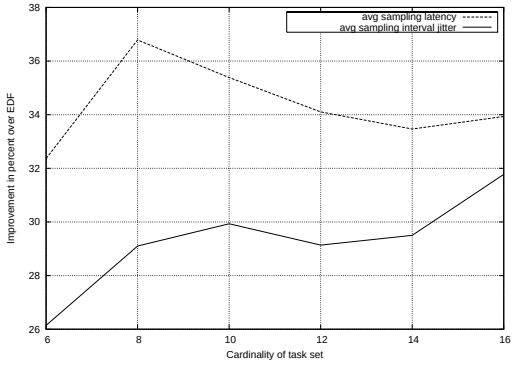

Figure 1. Average sampling latency and average sampling interval jitter with comparison to EDF for the best feasible policy found in the search space defined by equation 2.

On figure 1, one sees that for a particular application, improvements are always larger than 32% for average sampling latency and larger than 26% for average sampling interval jitter whatever the cardinality of the set of tasks. For example, the average improvement achieved for 10 tasks is 35% for average sampling latency and 30% for average sampling interval jitter.

Overall, the technique is efficient, even on heavily loaded systems (average load of 0.85 in our experiments) and the improvement over EDF for average sampling latency and average sampling interval jitter is really significant whilst always ensuring feasibility. Similar results, not shown here, were found for the average input-output latency and the input-output latency jitter.

6. Conclusion and future work

In this paper, we highlight a class of on-line scheduling algorithms that are both easy to implement and that can provide interesting performances for feasibility and, especially, for other application-dependent criteria. We propose an algorithm to compute worst-case response time bounds that is generic for all policies of the class. Experiments show that, in the context of computer-controlled systems where delays and jitters impact the performances of the control loop, well chosen policies can bring important improvements over plain EDF.

In the future, we intend to evaluate more precisely the impact of the scheduling policies using software tools such as TrueTime [6] or the tool described in [12], that allow to integrate delays induced by the scheduling in the control loops. It is also planed to experiment new search techniques for exploring the policy search space; preliminary experiments show that simple neighborhood techniques such hill-climbing are much more efficient than exhaustive search.

This work could be extended to other class of policies such as time-sharing policies (e.g. Round-Robin, Pfair). The main problem will be here to come up with a generic schedulability analysis.

References

[1] K. Altisen, G. Goessler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In *Proc. of the 20th IEEE Real-Time Systems Symphosium (RTSS 1999)*, 1999.

[2] K. Astrom and B. Wittenmark. *Computer-Controlled Systems*. Prentice Hall ISBN 0-13-168600-3, third edition edition, 1997.

[3] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari. Scheduling periodic task systems to minimize output jitter. In *Proc. of the 6th International Conference on Real-Time Computing Systems and Applications (RTCSA 1998)*, 1998.

[4] D. P. Bunde. SPT is optimally competitive for uniprocessor flow. *Information Processing Letters*, 90(5):233–238, 2004.

[5] A. Cervin. *Integrated Control and Real-Time Scheduling*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, 2003.

[6] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén. How does control timing affect performance? *IEEE Control Systems Magazine*, 23(3):16–30, 2003.

[7] A. Crespo, I. Ripoll, and P. Albertos. Reducing delays in rt control: the control action interval. In *Proc. of the 14th IFAC World Congress*, 1999.

[8] L. David, F. Cottet, and N. Nissanke. Jitter control in on-line scheduling of dependent real-time tasks. In *Proc. of the 22nd IEEE Real-time Systems Symposium (RTSS 2001)*, 2001.

[9] J. Goossens and P. Richard. Performance optimization for hard real-time fi xed priority tasks. In *Proc. of the 12th international conference on real-time systems (RTS 2004)*, 2004.

[10] M. Grenier and N. Navet. New preemptive scheduling policies for improving real-time behavior. To appear in IEEE ETFA conference, Catania, Italy, sep 2005.

[11] E. Grolleau, A. Choquet-Cheniet, and F. Cottet. Validation de systèmes temps réel à l'aide de réseaux de petri. In *Proc. of Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'98)*, 1998.

[12] F. Jumel, N. Navet, and F. Simonot-Lion. Evaluation de la qualité de fonctionnement d'une application de contrôle-commande en fonction de caractéristiques de son implantation. *to appear in Technique et Science Informatiques*, 2005.

[13] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard-real time environnement. *Journal of the ACM*, 20(1):40–61, 1973.

[14] J. Migge. *Scheduling under Real-Time Constraints: a Trajectory Based Model*. PhD thesis, University of Nice Sophia-Antipolis, 1999.

[15] N. Navet and J. Migge. Fine tuning the scheduling of tasks through a genetic algorithm: Application to posix1003.1b compliant os. In *Proc. of IEE Proceedings Software*, volume 150, pages 13–24, 2003.

[16] B. Wittenmark, J. Nilsson, and M. Törngren. Timing problems in real-time control systems. In *Proc. of the American Control Conference*, 1995.