Available online at www.sciencedirect.com

ScienceDirect

IFAC-PapersOnLine 49-6 (2016) 280–285

Teaching Task Scheduling as Multivariable Cascade Control

Alberto Leva [*] Federico Terraneo [**]

[*] Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (e-mail: alberto.leva@polimi.it)
[**] Research assistant at the Dipartimento di Elettronica, Informazione e Bioingegneria (e-mail: federico.terraneo@polimi.it)

Abstract: This paper presents a didactic activity belonging to a long-term project, aimed at complementing the culture of computer engineering students with a solid knowledge of systems and control theory and methods. The use of control to govern and optimise the behaviour of computing systems is felt in the computer engineering community as a necessity. The proposed activity – that refers to task scheduling – responds to this need, guiding the necessary cultural enrichment and avoiding possible errors and misinterpretations, so as to foster a deeper cooperation of the computer and the control communities.

Keywords: Control education, computing systems, scheduling algorithms, cascade control, discrete-time systems.

1. INTRODUCTION

The material and the activity presented in this paper belong to a long-term research and education project, aimed at control and control-based design of computing system components. On the research side, the main goal is to improve the performance and the sustainability of computing systems, see Leva et al. (2012). From the education standpoint, on which we here focus, the aim is to enrich the culture of computer engineering students with a purpose-specific, yet methodologically grounded knowledge of the systems and control theory.

The previous paper (Leva, 2015) provided an overview on the matter, also providing references to research works that ground the didactic activity, and are not repeated here for brevity. This work concentrates on the relevant problem of task scheduling, that is here formulated as a multivariable cascade control one in the discrete time domain. Synthesising the controller requires dealing with stability and performance in a linear but time-varying context, while realising the obtained control law involves nontrivial issues, particularly for the management of a variable task pool.

The activity here described is a good way for computer engineering students to get a firm grasp not only on some control techniques, but more in general, on how control ideas can help solve relevant problems if these are formulated properly in system-theoretical terms, and if control design is carried out on dynamic models instead of going directly to algorithms.

At the Politecnico di Milano, computer engineering students take a ten credits basic course on systems and control, in the second semester of the second BSc year. They can then choose some specialised courses, but none is explicitly focused on the application of systems and control methods to their core domain—whence the need for activities like the one presented herein. This year, the activity will be offered as part of a PhD course, with the purpose of finding the best organisation and communication strategy with the help of the students feedback. If the result is successful, the possibility can be considered to move the activity – together with the companion ones summarised in Leva (2015) – down to the MSc level, which is most likely their natural collocation.

The paper is organised as follows. Section 2 briefly reviews some related work, thereby further motivating the presented research. Sections 3 through 5 present the methodological part of the work, while Section 6 describes the implementation of the designed controller, which is of particular importance when dealing with a computer-centric audience. Section 7 deals with the organisation of the didactic activity, and the underlying pedagogy. Finally, Section 8 draws some conclusions, and sketches out future work.

2. RELATED WORK AND MOTIVATION

The topic “control of computing systems” has received an increasing attention in the last decades. The importance of the matter was very clear in the case where computing serves for control in the classical sense of the term, and an incorrect use of computational resources may cause control failures (Stankovic et al., 1999; Branicky et al., 2002).

Some time ago, however, control started gaining interest also for optimising the operation of computing systems that have nothing to do with regulating loops. Besides quite general and high-level treatises such as Shor et al. (2000), works like Abdelzaher et al. (2003), centred on Quality of Service issues, were in those years one of the most relevant trends, particularly for the management of distributed middleware (Li and Nahrstedt, 2001) and web servers (Robertsson et al., 2004).

In the following years, the idea of using control techniques to make computing systems “adaptive” – i.e., capable of maintaining certain properties in the presence of unforeseen ambient variations – gained importance (Diao et al., 2005; Huebscher and McCann, 2008; Salehie and Tahvildari, 2009; Filieri et al., 2011). At present, research extends to operating systems (Leva et al., 2012), resource management, especially with energy awareness (Beloglazov et al., 2012), cloud control (Klein et al., 2014; Ghezzi et al., 2015), self-adaptive software (De Lemos et al., 2013; Arcelli et al., 2015), and more; reviews and comparisons of alternative approaches to so vast a field can be found in works like Maggio et al. (2012) and Patikirikorala et al. (2012).

Moving to the scope of this paper, the "control of computing systems" topic has nowadays started the transition from a research only to a teaching subject. There are books suitable for such a purpose (Hellerstein et al., 2004; Janert, 2013), and some include a "minimal control course" targeted to a computer engineer reader. Quoting from the presentation of Janert (2013), "feedback is ideal for controlling large, complex systems, but its use in software engineering raises unique issues; this book provides basic theory and lots of practical advice for programmers with no previous background in feedback control".

This situation requires attention from the control community. In the absence of a solid pedagogy, well grounded on the systems theory, there is the risk that control is viewed basically as "yet another library of algorithms". For example, most computer-centric works on control introduce the basic regulator structures like the PID, but hardly any mention is made on selecting model- or problem-specific control laws, let alone on *structuring* a control scheme. More in general, looking at control with just the mentality of a programmer, may lead to miss the real cultural treasure, i.e., the capability of designing and assessing based on dynamic models.

Despite choosing the best way to share control culture with computer scientists is an open challenge under several viewpoints, no doubt computer people are getting really interested in control, and students in that domain need educating properly on this subject. Thus, in the opinion of the authors, we definitely must be there; the activity presented in the following is an attempt of providing such a presence.

3. MODEL OF THE SYSTEM UNDER CONTROL

A pool of task is to be scheduled on a single processor (CPU). We assume a preemptive context, which means that the scheduler can interrupt the running task and give the CPU to another one. The pool of task is not constant, as some tasks can terminate and leave the system, while new ones can arrive.

Tasks can be classified based on their desired use of the CPU. Some tasks are *interactive*: they need to respond to the user fast enough, thus to receive the CPU frequently enough. Others are *batch*: these just have to carry out a given workload within a deadline, while the detailed history of their CPU use is not relevant. Others again are *periodic*, and can be viewed as repeating batch ones. Others are *event-awakened*, like for example the driver of a mouse; these need the CPU as quickly as possible when the triggering event occurs, but differently from interactive ones, not on a regular basis. In a nutshell, and informally, the scheduler has to maximise the CPU use by the pool, while satisfying the desires of each task as much as possible.

Since pool variations are sporadic with respect to the scheduling time scale, one can first address the problem for a constant pool, and then manage task arrivals and terminations. Writing an adequate model is not trivial for the students. Computer engineering ones typically find it hard to isolate the controlled phenomenon – tasks accumulate CPU time "more or less" as dictated by the scheduler – and often come up with queue networks or analogous ideas. Students more acquainted to physics in the strict sense of the term, on the other hand, easily get lost in identifying the quantities themselves that describe the system.

Coming to the proposed solution, we preliminarily define the *round* as the time between two subsequent scheduler interventions, and we also define (the bold face denotes vectors and matrices)

- $\tau_\mathbf{t}(k) \in \mathbb{R}^N$, the CPU times actually used by the tasks in the k-th round, measured after their execution;
- $\tau_\mathbf{r}(k) \in \mathbb{R}$, the time duration of the k-th round;
- $\rho_\mathbf{t}(k) \in \mathbb{R}^N$, the times to completion (remaining needed CPU times) of the tasks at the beginning of the k-th round ($+\infty$ for the task that do not have a duration);
- $n(k) \in \mathbb{N}$, the number of tasks that the scheduler considers for activation at the k-th round;
- $\mathbf{b}(\mathbf{k}) \in \mathbb{R}^{n(k)}$, the *bursts*, i.e., the CPU times allotted to the tasks at the beginning of the k-th round;
- $\delta\mathbf{b}(k) \in \mathbb{R}^{n(k)}$, any disturbance (voluntary CPU yield, preemption interrupt latency, and so on) making $\tau_\mathbf{t}(k)$ differ from $\mathbf{b}(k)$.

Given this, and denoting by t the time since the system initialisation, an adequate model reads

$$\begin{cases} \tau_\mathbf{t}(k) = \mathbf{S}_\sigma(k-1)\cdot\mathbf{b}(k-1)+\delta\mathbf{b}(k-1) \\ \tau_r(k) = \mathbf{1}_{1\times N}\cdot\tau_\mathbf{t}(k-1) \\ \rho_\mathbf{t}(k) = \max\left(\rho_\mathbf{t}(k-1)-\mathbf{S}_\sigma(k-1)\mathbf{b}(k-1)-\delta\mathbf{b}(k-1),0\right) \\ t(k) = t(k-1)+\tau_r(k) \end{cases} \tag{1}$$

where $\mathbf{S}_\sigma(k) \in \Sigma$ is an $N \times n(k)$ switching matrix, with elements equal to zero or one, that determines which tasks are considered in each round (notice that this allows extensions to multiple CPUs). By choosing $n(k)$ and/or $\mathbf{S}_\sigma k)$, (1) can be made to describe many well known scheduling policies. To give just a few examples,

- $n = 1$, a N-periodic $\mathbf{S}_\sigma(\cdot)$, $\mathbf{S}_\sigma(\mathbf{k}) \neq \mathbf{S}_\sigma(k-1), 2 \le k \le N$ and a constant $b(k)$, produce Round Robin;
- $n = 1$ and $\mathbf{S}_\sigma(\cdot)$ chosen to assign the CPU to the task with the minimum row index and a positive $\rho_\mathbf{t}$, give First Come First Served;
- $n = 1$ and a $\mathbf{S}_\sigma(\cdot)$ selecting the task with the minimum $\rho_\mathbf{t}$, yield Shortest Remaining Time First.

Other policies can be described, and it is interesting for the students to distinguish open-loop ones, like Round Robin, from closed-loop ones. However, once a model is available for the system in the absence of control – which is proven by its ability to represent open-loop policies – the question arises, why the said model cannot be used to synthesise a scheduling policy in the form of a (MIMO) controller. A possible solution is discussed in the following.

4. CONTROL STRUCTURING

Observe that all the existing policies just mentioned, and virtually any one used in operating systems, have $n = 1$, i.e., have the scheduler gain control after the execution of each task. This is not necessary: one can determine all the $\mathbf{b}(k)$ vector